

## **NASA Contractor Report 181742**

# **The CSM Testbed Matrix Processors Internal Logic and Dataflow Descriptions**

(NASA-CR-181742) THE CSM TESTBED MATRIX  
PROCESSORS INTERNAL LOGIC AND DATAFLOW  
DESCRIPTIONS (Lockheed Missiles and Space  
Co.) 91 p CSCI 20K

N89-14474

Unclas  
G3/39 0185076

**Marc E. Regelbrugge and Mary A. Wright**

**Lockheed Missiles and Space Company, Inc.  
Palo Alto, California**

**Contract NAS1-18444**

**December 1988**



National Aeronautics and  
Space Administration

**Langley Research Center**  
Hampton, Virginia 23665-5225

11

12

13

14

15

16

# The Computational Structural Mechanics Testbed Matrix Processors

## Internal Logic and Dataflow Descriptions

- 1.0 Introduction
  - 1.1 Overview
  - 1.2 Definitions and Notations
  - 1.3 Testbed Sparse Matrix Data Structure
- 2.0 System Matrix Processors
  - 2.1 TOPO
  - 2.2 K
  - 2.3 INV
  - 2.4 SSOL
- 3.0 Utility Processors
  - 3.1 AUS
- 4.0 References



### 1.0 Introduction

This report constitutes the final report for subtask 1 of Task 5 of NASA Contract NAS1-18444, Computational Structural Mechanics (CSM) Research. This report contains a detailed description of the "coded" workings of selected CSM Testbed matrix processors (i.e., TOPO, K, INV, SSOL) and of the arithmetic utility processor AUS. These processors and the current sparse matrix data structures are studied and documented. Items examined include: details of the data structures, interdependence of data structures, data-blocking logic in the data structures, processor data flow and architecture, and processor algorithmic logic flow.

THIS PAGE LEFT BLANK INTENTIONALLY.

## 1.1 Overview

This document describes details of the coded workings of the CSM Testbed matrix processors TOPO, K, INV, SSOL and the utility processor AUS. The purpose of this description is twofold:

- 1) To provide a clear description of the internal logic and data flows and of the user and database interface requirements of these processors so as to enable the straightforward and accurate modification of these processors to enhance the analytical capabilities of the Testbed.
- 2) To lend insight into the necessary functionality and key architectural features of these processors in order to guide future development of matrix-algebra oriented software along functionally rational and utility-oriented lines.

The centerpiece of the present description is the set of logic flowcharts developed for key subroutines of each of the system matrix processors TOPO, K, INV, SSOL and AUS. These charts, along with commented FORTRAN source code also produced under this CSM activity, allow a knowledgeable code developer to deduce the logic of the documented routine and to locate quickly particular sections of code either to be modified or to serve as examples for further explanation. The reader should make a special note of the symbiotic relationship between the logic flowcharts and the commented FORTRAN source code, *i.e.*, both need to be reviewed together in any serious study of these modules' internal workings. This is because the details of FORTRAN programming practices are frequently best understood upon careful study of the code itself. Many of these details are simply too intricate and specialized to be well described by flowcharts or technical English. The programming described here is not for beginners. On the other hand, the logic flowcharts provide a more general view of the processor modules from which the reader may deduce modules' operations in the context of the architecture of the entire processor. The logic flowcharts allow one to examine program logic flow on a level above that which is possible in study of a one-page program listing.

Each matrix processor described in Chapter 2 is also fully documented as to processor and/or subprocessor name, function, user inputs (RESET and other commands), input and output database datasets, and internal core allocation practices. In addition, the data storage structure of the Testbed sparse matrix is documented in §1.3. The interested reader is referred to the references listed in §4.0 for additional information. We wish to note that the activities undertaken to provide the present description were greatly simplified by the prior effort expended in the production of the referenced documents (§4.0).

THIS PAGE LEFT BLANK INTENTIONALLY.



## 1.2 Definitions and Notation

The following key is helpful in the interpretation of the logic flowcharts.

- lowercase - general information
- UPPERCASE - user commands (*e.g.*, RESET, INLIB) or generic entities (*e.g.*, KMAP buffer)
- BOLDFACE** - FORTRAN-identifiable quantities (*e.g.*, variable names, subroutine names, statement labels, etc.)
- SLANTED - database entities. Input datasets appear *above* the dataflow lines whereas output datasets appear *below* the dataflow lines
- \* (asterisk) - generic wild-card match

In the accompanying text, FORTRAN-identifiable quantities appear in typewriter font. Like many FORTRAN compilers, no distinction is expressly drawn between subroutine, function, and variable names. Common block names are written with leading and trailing slashes, like /THIS/. In keeping with FORTRAN-IV conventions, and since the vast majority of the source code described herein appears in uppercase, FORTRAN entities appear only in uppercase.

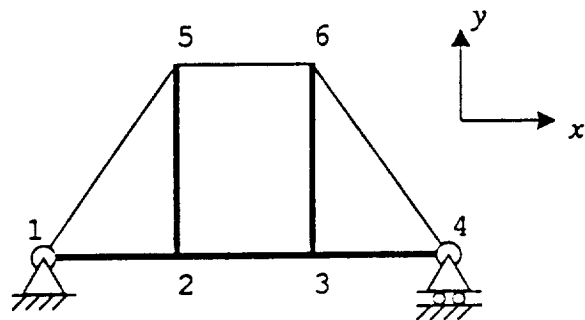
THIS PAGE LEFT BLANK INTENTIONALLY.

### 1.3 Testbed Sparse Matrix Data Structure

This section describes the data storage structures of the Testbed sparse matrix. Throughout this section, an example model depicted in figure 1 will be referenced. This example is a simple finite-element model comprising five beam elements, two triangular plate elements and one quadrilateral plate element. For purposes of illustration, all six nodes are assumed to have six active degrees-of-freedom (d.o.f.), providing a total of 36 d.o.f. in the entire model. The nodal d.o.f. are numbered in the conventional sense; one through six being associated with translational motions in the  $x$ ,  $y$ , and  $z$  directions and rotations about the  $x$ ,  $y$ , and  $z$  axes, respectively. Note that the d.o.f. associated with all translations at node 1 and with  $y$  and  $z$  translations at node 4 are suppressed by support boundary conditions.

Example Problem Model:

6 nodes  
6 d.o.f./node



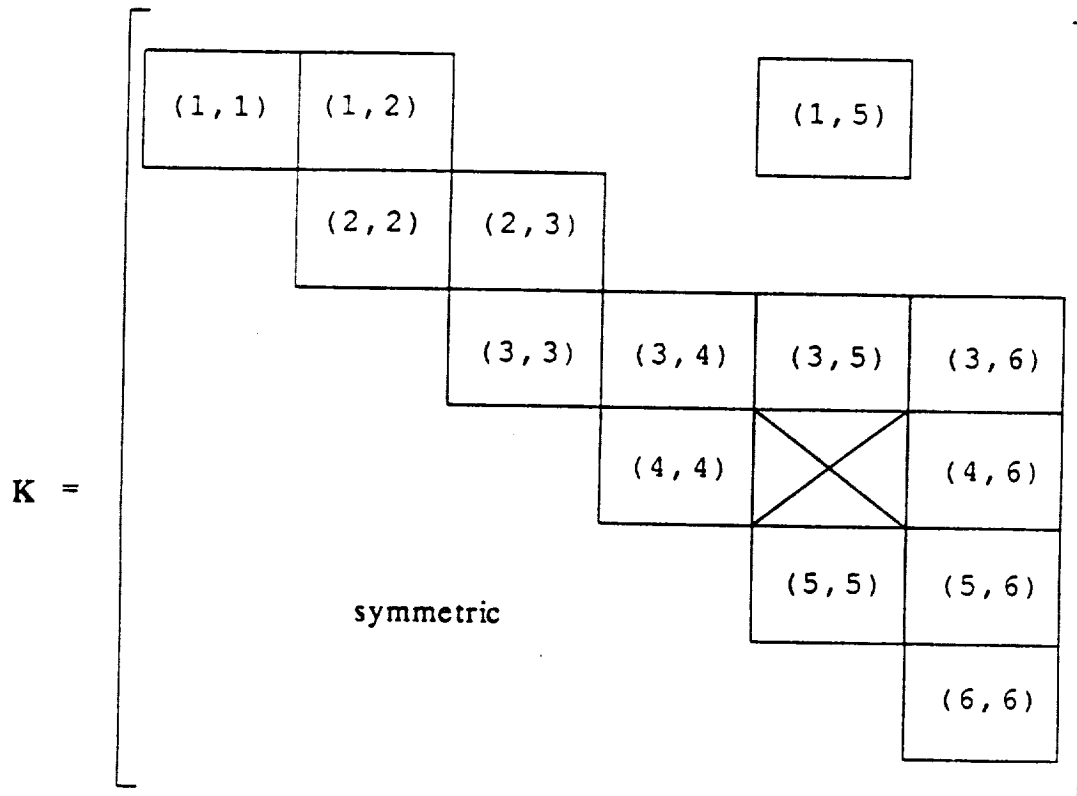
Element #	Element type	Connected Nodes
1	beam	1, 2
2	beam	2, 3
3	beam	3, 4
4	beam	2, 5
5	beam	3, 6
6	plate	1, 2, 5
7	plate	3, 4, 6
8	plate	2, 3, 6, 5

Figure 1 Example Finite Element Model.

The Testbed sparse matrix data structure is a nodal-block oriented scheme for storing the elements of the upper triangle of a sparse, symmetric system matrix. The Testbed sparse matrix is stored in one of two forms depending on whether the matrix has been factored. Figure 2 shows the logical structure of the unfactored Testbed sparse matrix using the interrelationships of the nodal-block submatrices for the example problem. Note that each box like

(1,2)

denotes a 6 by 6 nodal-block submatrix connected to the two nodes listed in parentheses inside the box. In the example above, the block indicated contains the coupling contributions from nodes 1 and 2. In the example problem, elements 1 and 6 contribute to this nodal-block submatrix (1,2). In the example matrix of figure 2, the only block whose terms are present in the factored matrix but absent in the unfactored matrix is marked with a large "X."



Key:

(1,5)

Indicates 6 by 6 nodal-block submatrices. In the case at left, the submatrix due to element connectivity between nodes 1 and 5 is depicted.



Indicates nodal-block submatrix that is not present in the model stiffness, but will fill in during factoring.

Figure 2 Sparse Matrix Nodal-Block Structure.

Both factored and unfactored Testbed sparse matrices are stored in a blocked, partitioned record scheme. Individual records are of constant length and contain both indexing data and matrix value data. The indexing data are useful only as integer type, but are stored physically in the unfactored matrix structure in the same datum precision as the

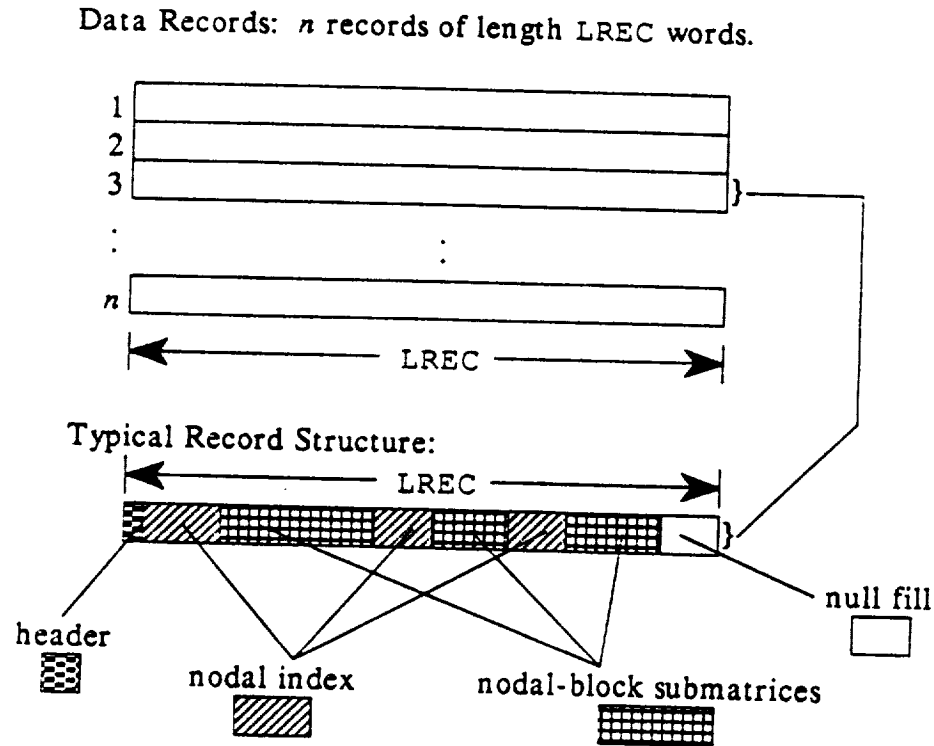
terms of the matrix itself. In the factored matrix structure, however, the indexing data are stored as integer type regardless of the datum precision of the matrix values. The record partitions differ in detail between the factored and unfactored matrix structures, owing primarily to the incorporation of constraint (d.o.f. suppression) information into the factored matrix structure.

The record partitioning scheme and record contents for unfactored stiffness matrix of the figure 1 example problem are presented in figures 3(a) and 3(b). To make the substance of figure 3(b) more illustrative, a record length (LREC) of 384 words has been chosen. The fundamental unit of information in the record partitioning scheme is the nodal-block subrecord, which comprises nodal index information and all nodal-block submatrices that contribute to the rows assigned to the diagonal-block node in the upper triangle of the system matrix. The first node listed in the nodal index is referred to as the diagonal-block node since its nodal block appears on the diagonal of the system matrix. The nodal index information contains the number of nodal-block submatrices present in the subrecord (for the current diagonal-block node) and the node numbers associated with the columns of these nodal-block submatrices. The size of each nodal-block submatrix is the square of the number of nodal d.o.f. *not* constrained on the START card in the TAB Testbed modeling processor.

Note that the records are partitioned so that complete nodal subrecords are contained within one record, i.e., the matrix information associated with a nodal-block row of the matrix is not allowed to span record boundaries. Thus, the record size is used only as a data manager parameter, and transmits no specific information about the matrix itself, or how the record partitions are to be interpreted. All interpretive information is encountered sequentially as the record is processed from the first word through the LREC<sup>th</sup> word.

The record partitioning scheme and record contents for the factored stiffness matrix of the figure 1 example problem are presented in figures 4(a) and 4(b). For purposes of illustration, a record length (LRA) of 384 words was chosen for the detail of the record contents in figure 4(b), and only the first record is shown. The subscripts of the  $D^{-1}$  and  $L$  terms in figure 4(b) refer to d.o.f. numbers, assigned sequentially in groups of six to each node.

As in the unfactored matrix structure, nodal subrecords in the factored matrix are not allowed to span record boundaries. Unlike the unfactored matrix structure, constraint data associated with nodal d.o.f. suppressions is included in the matrix data records. The factored matrix rows corresponding to suppressed d.o.f. are not included in the data. A map is provided at the beginning of the nodal subrecord to indicate the active d.o.f., as an indexed subset  $(1, \dots, n)$  of the d.o.f. not constrained on the START card in TAB, for the current diagonal node. An interesting observation is that the factored matrix data cannot be decoded completely without additional information about the number of d.o.f. per node in the finite element model, and which nodal d.o.f. are potentially active. In the Testbed, this information is obtained from a modeling summary dataset JDF1.BTAB.1.8.






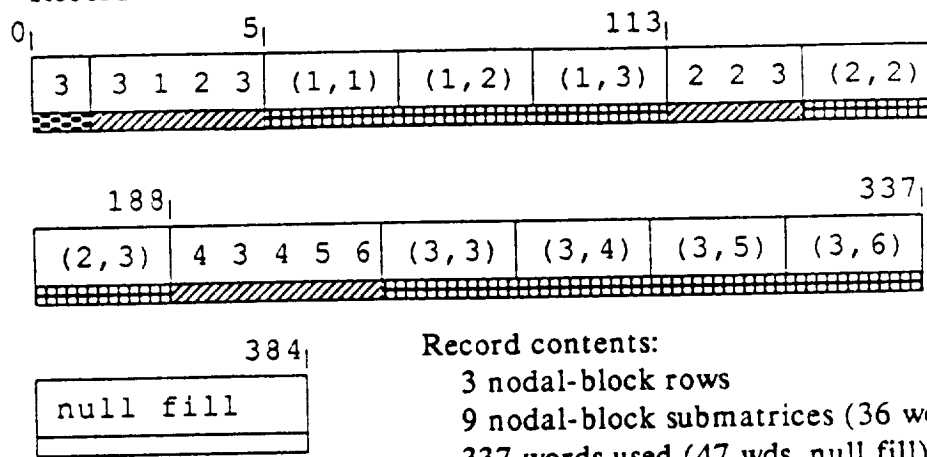
Subrecord	Key	Contents
header		Number of nodal-block rows in the upper triangle of the system matrix contained in this record.
nodal index		Number of nodes contributing to the nodal-block submatrices in this row and the numbers of these nodes.
nodal-block submatrices		6 by 6 submatrices of matrix coefficients in the rows of the upper triangle of the matrix connected to the nodes listed in the nodal index.

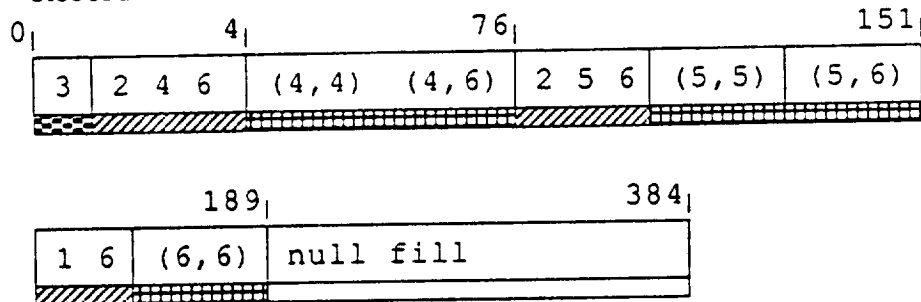
Figure 3(a) Record Partitioning Scheme for Testbed Sparse Matrix.

Record 1:






Record contents:  
 3 nodal-block rows  
 9 nodal-block submatrices (36 wds. ea.)  
 337 words used (47 wds. null fill)

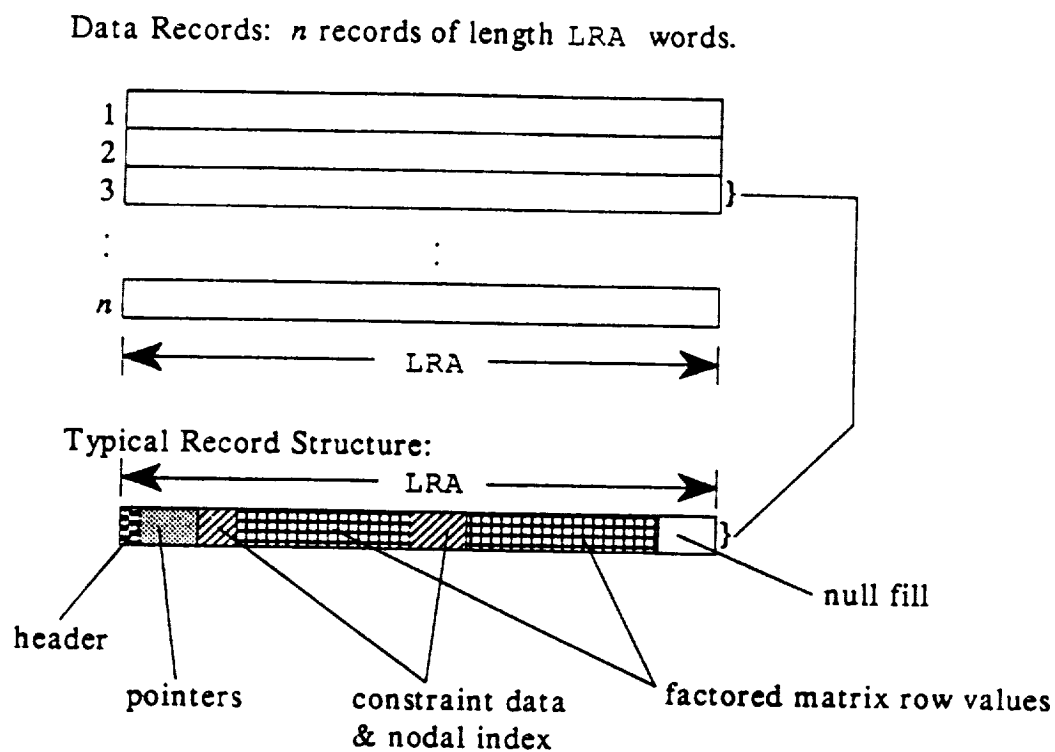
**Record 2:**



Record contents:  
 3 nodal-block rows  
 5 nodal-block submatrices (36 wds. ea.)  
 189 words used (195 wds. null fill)

Subrecord	Key	Contents
header		Number of nodal-block rows in the upper triangle of the system matrix contained in this record.
nodal index		Number of nodes contributing to the nodal-block submatrices in this row and the numbers of these nodes.
nodal-block submatrices		6 by 6 submatrices of matrix coefficients.

**Figure 3(b)** Sparse Matrix Record Contents for Example Problem.







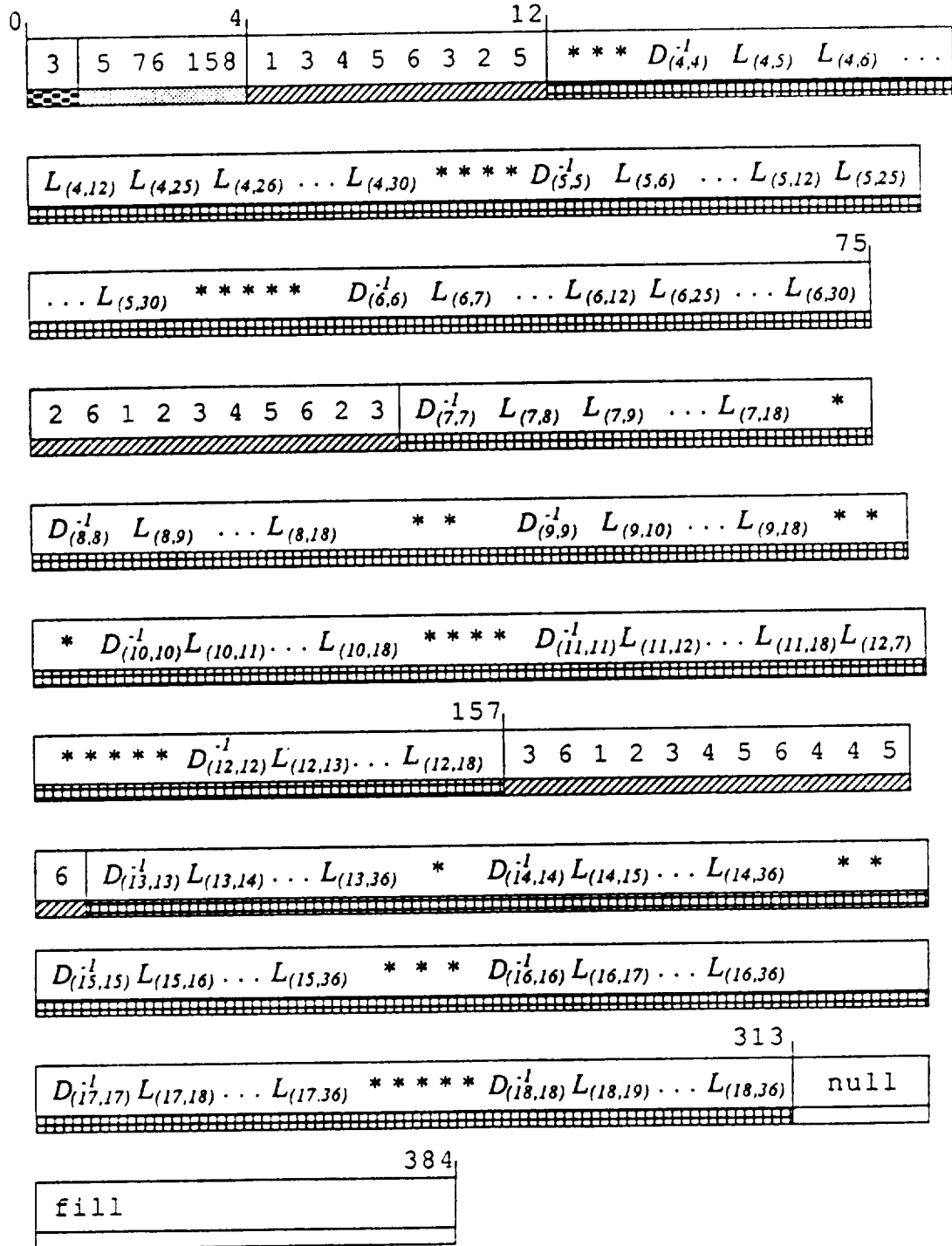
Subrecord	Key	Contents
header		Number of nodal-block subrecords in this record.
pointers		Physical (word) pointers to start of each subrecord in this record.
nodal index		Number of active d.o.f. and d.o.f. indices for current node, number of nodal-block row submatrices to follow and the numbers of the nodes associated with these row submatrices.
factored row nodal-block submatrices		Nodal-blocks of rows of terms in the upper triangle of the factored matrix for each active d.o.f. of the current node.

Figure 4(a) Record Partitioning Scheme for Testbed Factored Matrix.





\* - Denotes an unused, subdiagonal entry.

Figure 4(b) Record Contents for Example Problem's Testbed Factored Matrix.

As an aside, one should note that the rather elaborate record partitioning schemes used for the Testbed matrices are byproducts of the architecture of the underlying data management system (DAL). Three DAL features in particular are responsible for the original Testbed design choices to place indexing and matrix values data side-by-side in the data records and to break the matrix storage into fixed-length segments (*i.e.*, records). These are:

- 1) DAL is a singly indexed hierarchical data manager, so to group data in logically related sets frequently requires the use of inhomogenous data records within a single dataset.
- 2) DAL handles datasets containing fixed-length records only. Different records in the same dataset cannot have different lengths.
- 3) DAL is sector (physical disk block) addressable at the finest granularity. Thus, it is required that integral numbers of disk blocks be read or written through DAL. For practical core memory limitations and the most efficient use of disk space, the large matrices are blocked into records that are sized to integral disk block sizes.

The pertinent observation to be made at this point is that the structure of matrix data is influenced not only by the structure of the matrix itself (in terms of zero and nonzero coefficients), but also by the operational characteristics of auxiliary data management software. Herein lies the most intimate connection between the algebraic and data descriptions of the system matrix.

## 2.0 System Matrix Processors

The four processors described in this Chapter are associated with assembly, factorization and solution of sparse-matrix format system matrix equations. These processors are:

- **TOPO** – Element Topology Analyzer. Processor TOPO analyzes element interconnection topology and creates datasets which guide the assembly and the factorization of system matrices.
- **K** – The System Stiffness Matrix Assembler. Processor K assembles unconstrained system matrices in the standard sparse-matrix format. If the appropriate elemental arrays have been formed, processor K may be used to assemble either the system material stiffness or the system geometric stiffness matrix.
- **INV** – Sparse-Matrix Format Factoring Processor. Processor INV factors the assembled sparse-matrix format system matrices.
- **SSOL** – Static Solution. Processor SSOL performs forward reduction and back substitution using the factored system matrix, to obtain the static displacements and reactions due to applied loads. Loads are combined from both processor EQNF and processor AUS.

THIS PAGE LEFT BLANK INTENTIONALLY.

## 2.1 Processor TOPO

### 2.1.1 GENERAL DESCRIPTION

Processor TOPO performs topology analysis and constructs the maps used in the assembly and factorization of Testbed sparse format system matrices.

The system topology maps, KMAP..*nsubs.ksize* and AMAP..*ic2.isize* are designed to facilitate system matrix assembly (using processor K) and factorization (using processor INV). TOPO handles much of the local memory management for subsequent executions of K and INV and communicates this information using the blocked, map data structures. As such, the maps are purely internal data, and are not normally for Testbed user perusal.

### 2.1.2 PROCESSOR SYNTAX

This processor follows Testbed command syntax and data management conventions as described in Reference 2.

#### 2.1.2.1 Processor Resets

<i>Argument</i>	<i>Default</i>	<i>Meaning</i>
BLIB	1	Input library number
LRKMAP	896	Length of KMAP.. <i>nsubs.ksize</i> records
LRAMAP	1792	Length of AMAP.. <i>ic2.isize</i> records
LR7	896	Length of records in scratch library number 26
MAXSUB	1400	Max. number of submatrices used during any stage of assembly or factoring
ILMAX	0	Max. nodal connectivity allowed. If not reset, it will be calculated based on MAXSUB
LAPROX	0	Estimated number of elements. If not reset, it will be calculated.
SA	0	Diagnostic print flag – print almost everything
PRTKMAP	0	KMAP.. <i>nsubs.ksize</i> print flag
PRTAMAP	0	AMAP.. <i>ic2.isize</i> print flag
PRT7	0	Print scratch library number 26 records
HLIB	1	KMAP.. <i>nsubs.ksize</i> destination library
ILIB	1	AMAP.. <i>ic2.isize</i> destination library

### 2.1.3 SUBPROCESSORS AND COMMANDS

Not applicable.

## 2.1.4 PROCESSOR DATA INTERFACE

### 2.1.4.1 Processor Input Datasets

- JSEQ.BTAB.2.17 (optional)
- ELTS.NAME
- DEF.xxxx.itype.nnod
- xxxx.EFIL.itype.nnod
  - xxxx is the element type name
  - itype is the element type number
  - nnod is the number of joints per element

### 2.1.4.2 Processor Output Datasets

- KMAP..nsubs.ksize
  - nsubs is total number of nodal submatrices in the K.SPARE.jdf2
  - ksize is the minimum required size (in submatrix units) of the assembly workspace (ksize less than MAXSUB)
- AMAP..ic2.isize
  - ic2 is a measure of the number of submatrix computations needed to factor the matrix
  - isize is the number of submatrices needed in core during matrix factoring

### 2.1.4.3 Processor Scratch Libraries

- Two workspace libraries (default: L25 & L26)

## 2.1.5 PROCESSOR LOGIC FLOW

Figures 5 through 9 contain flowchart diagrams for the TOPO top-level subroutine, TOPOEX, which directs TOPO computations, subroutine ELSORT which sorts element connectivity data into scratch data structures, and subroutines KMAP and PRECON which produce the system matrix map (KMAP..nsubs.ksize) and the factored matrix map (AMAP..ic2.isize), respectively. Other supporting subroutines, ELCON and ELSUB serve to manage working arrays and pointers for nodal connectivity mapping (CONROW, CONECT), and allocation pointers and block availability arrays (BLOCK, AVAIL) for the submatrix-block assembly operation. These arrays are volatile local memory and are used on a demand basis. ELCON manages CONROW and CONECT. ELSUB manages BLOCK and AVAIL.

The first task undertaken in TOPO is to sort elements into groups associated with each node in the problem. Elements are assigned to nodal groups if they reference the given node *and* at least one higher-numbered node (in the sense of the elimination sequence). This sorting is accomplished in ELSORT in two phases: a coarse sort and a fine sort. The coarse

sort stores certain element information into records in scratch file L25 (NU6) associated with the lowest-referenced element node. These scratch records are processed in element order, and are random with respect to nodal order. They are collected into nodal order in the final sort and written to a blocked data structure on library L26 (NU7).

The only essential differences between KMAP..nsubs.ksize and AMAP..ic2.isize, once the nodal connectivity has been determined, are that KMAP..nsubs.ksize holds some element data and that the AMAP..ic2.isize must account for nodal blocks which fill-in during factoring. The mapping logic in subroutines KMAP and PRECON is virtually identical with the exception of the DO 1300 loop to account for fill-in blocks in PRECON.

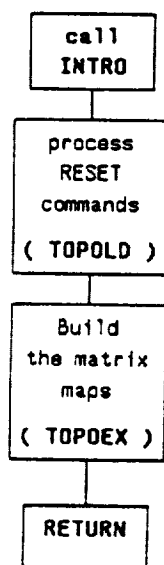


Figure 5 TOPO main program logic flowchart.



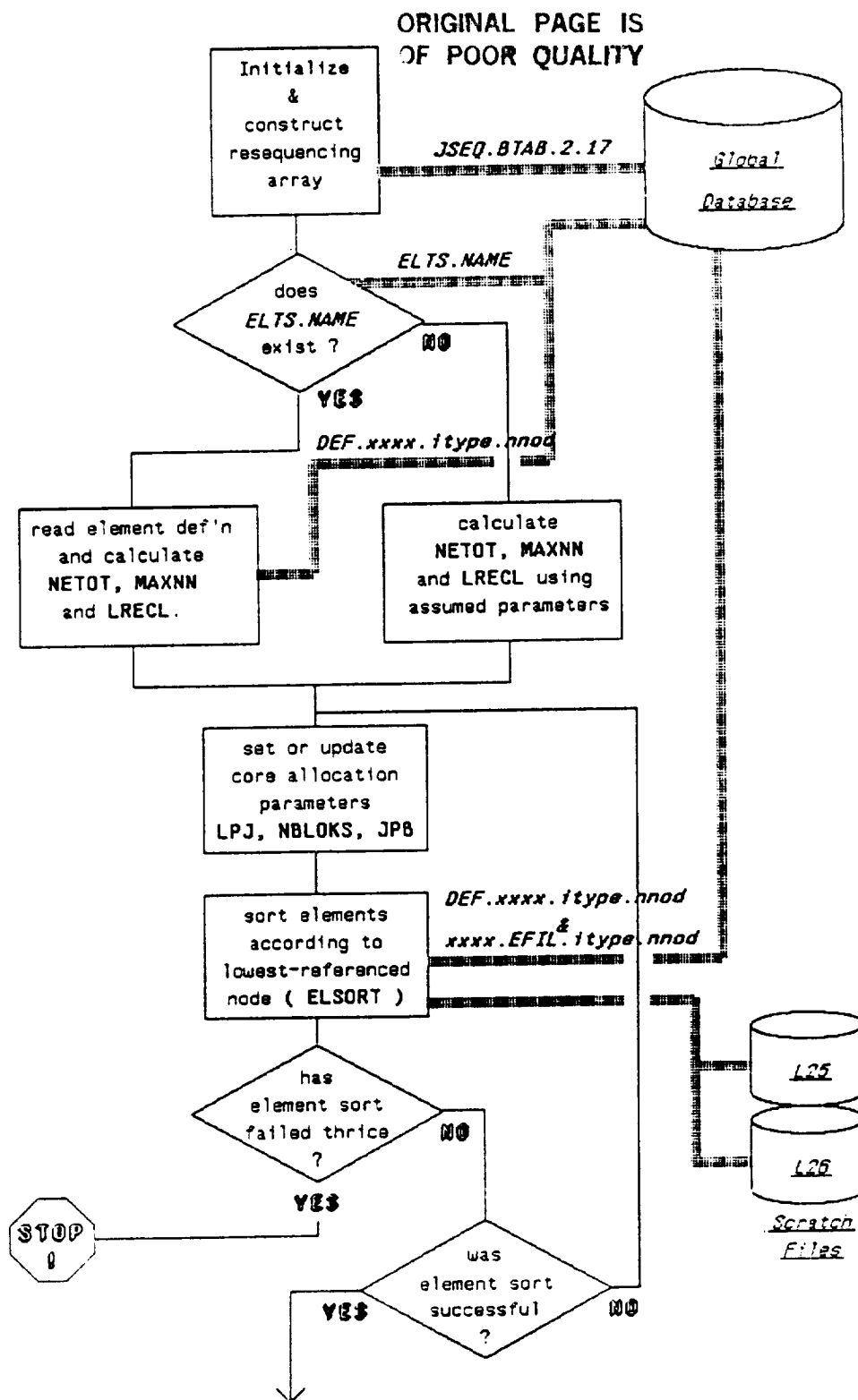


Figure 6 TOPOEX logic flowchart.

ORIGINAL PAGE IS  
OF POOR QUALITY

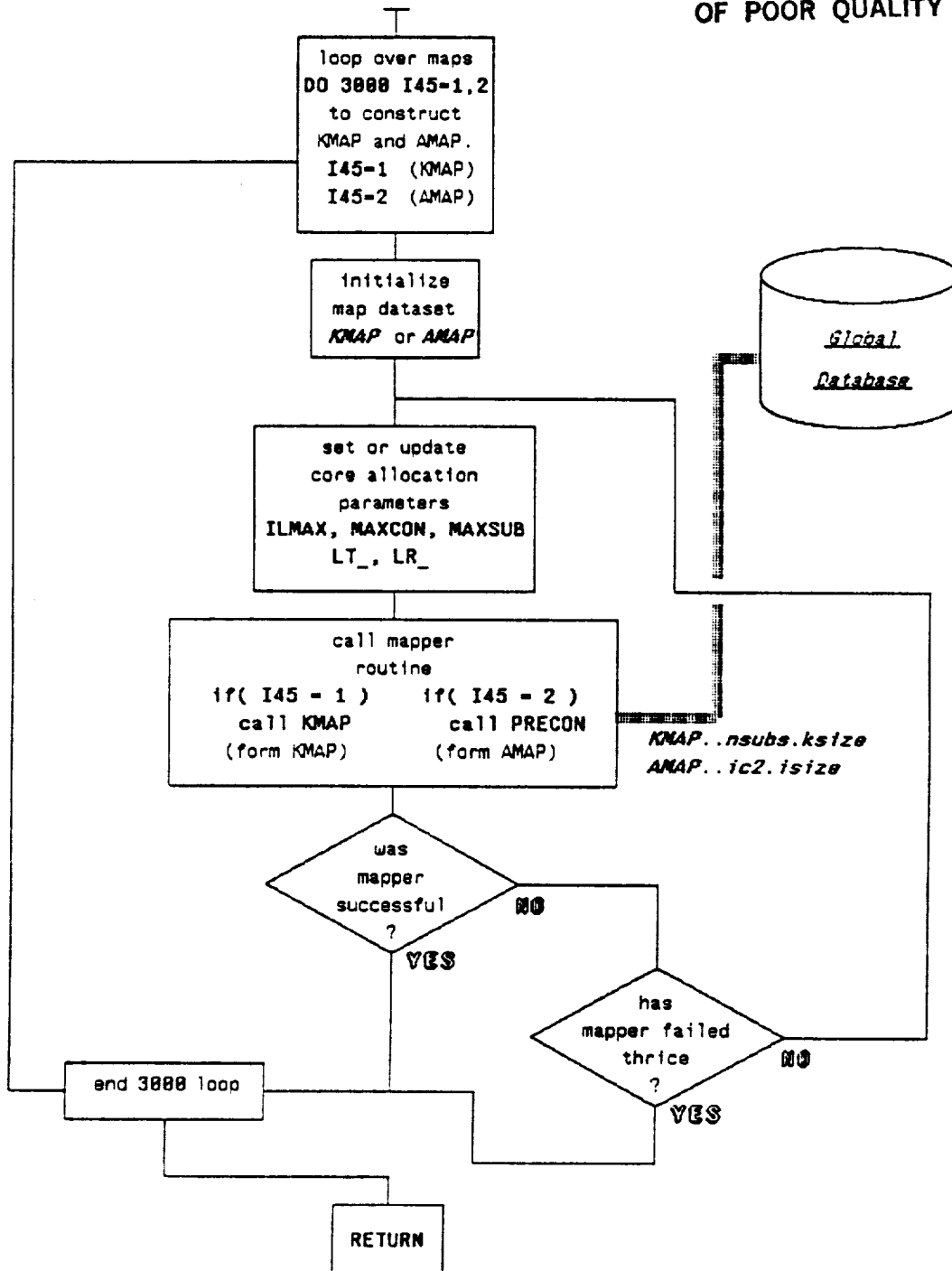


Figure 6 Concluded.

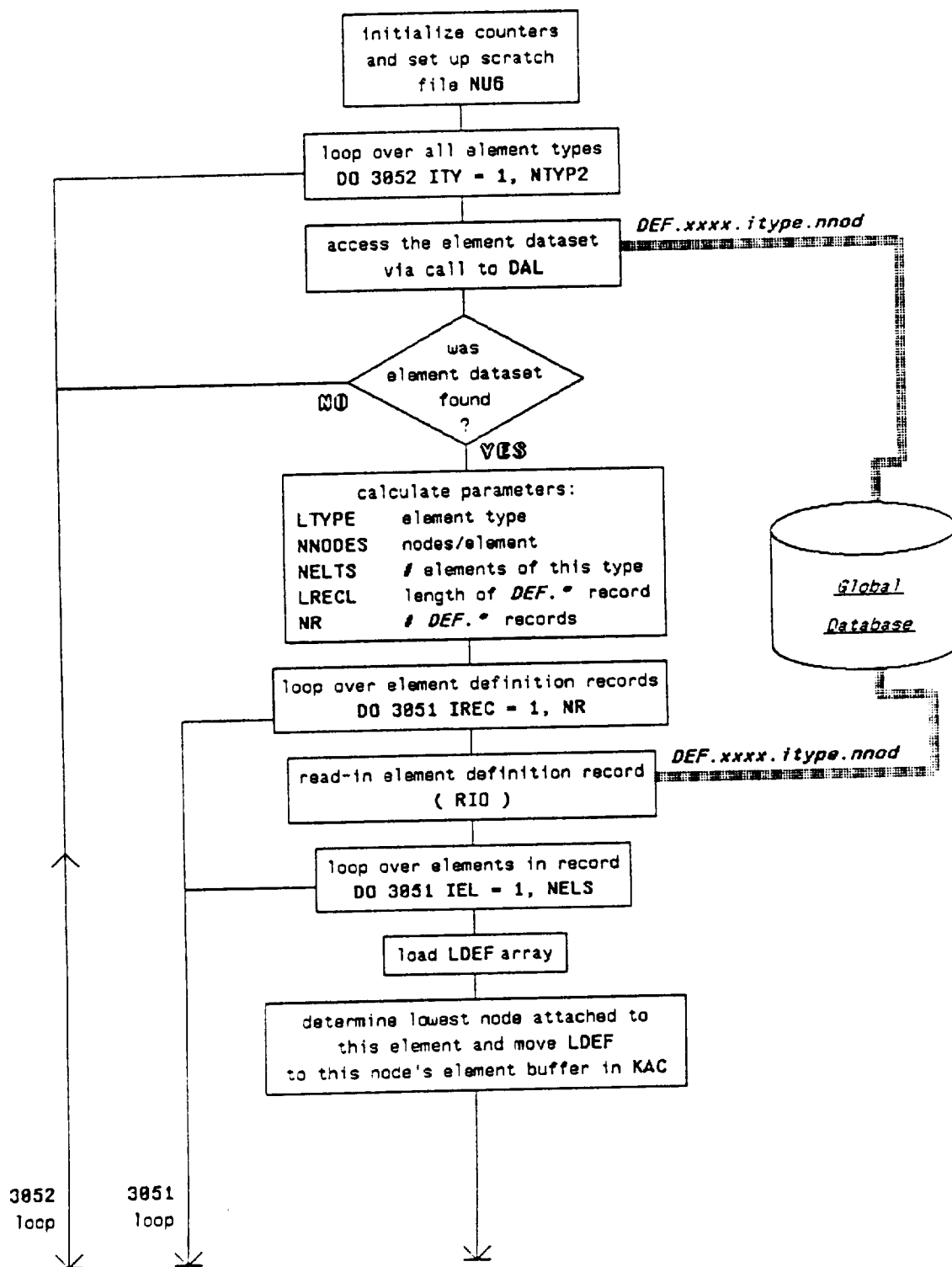
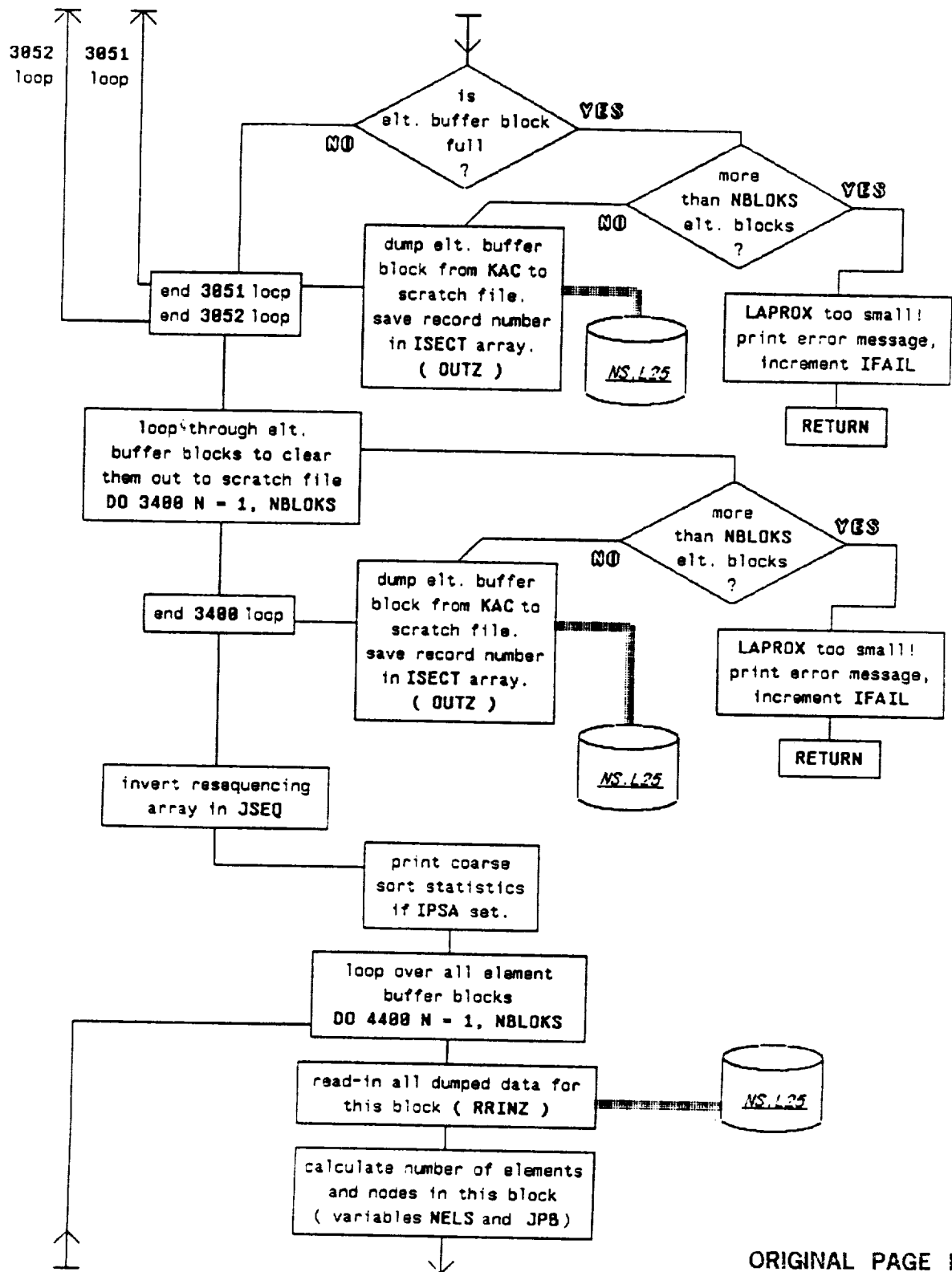


Figure 7 ELSORT logic flowchart.



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 7 Continued.

ORIGINAL PAGE IS  
OF POOR QUALITY

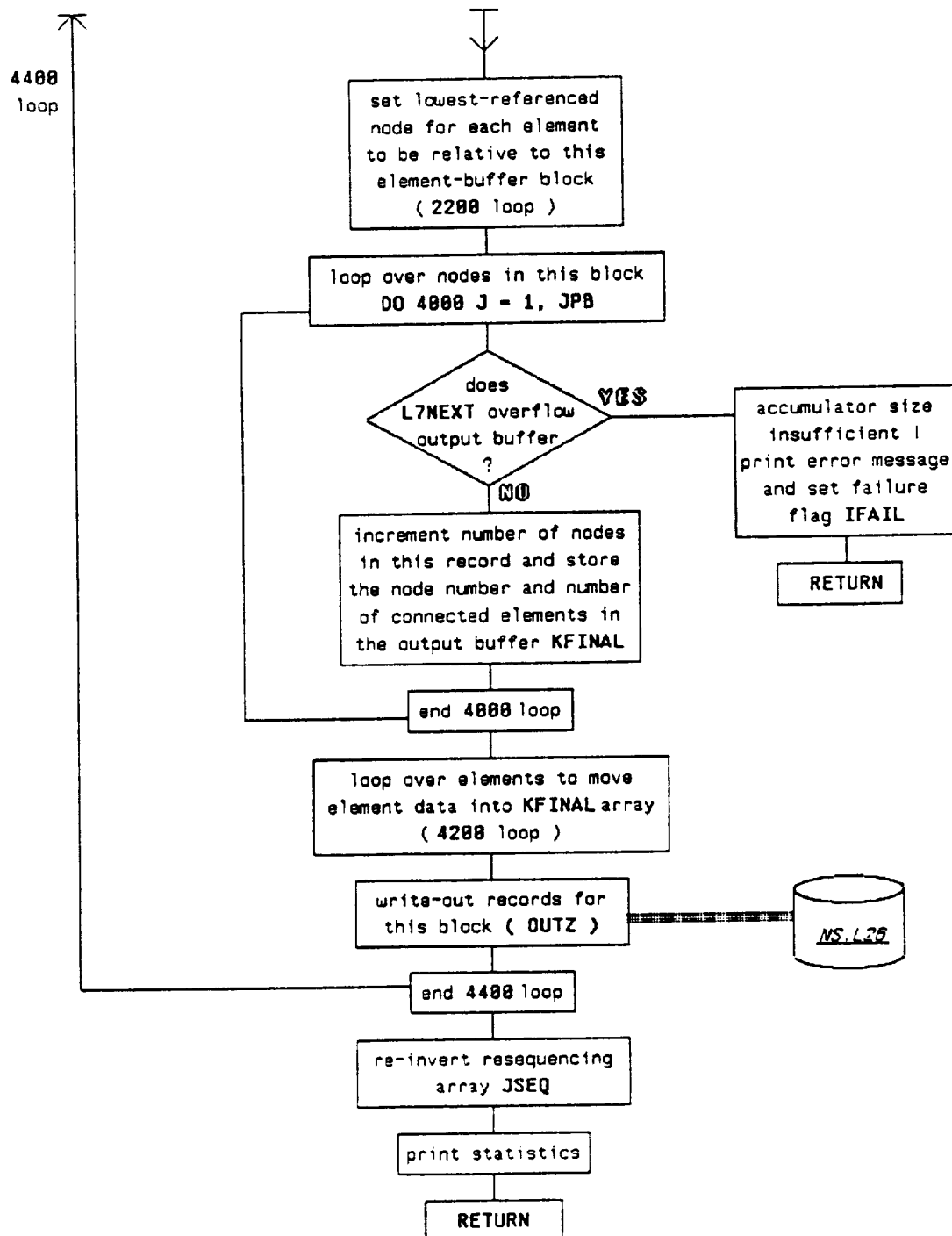


Figure 7 Concluded.

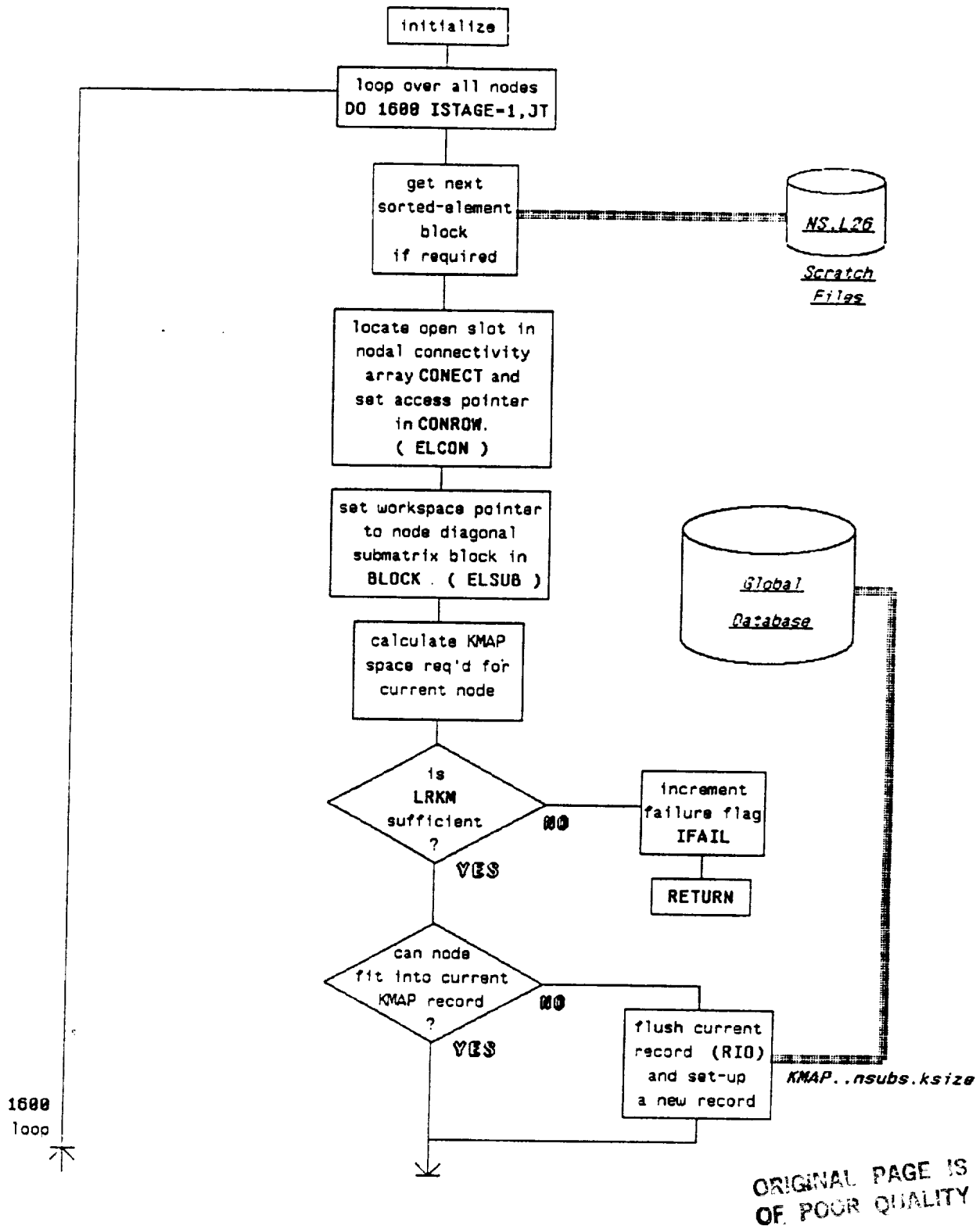


Figure 8 KMAP logic flowchart.

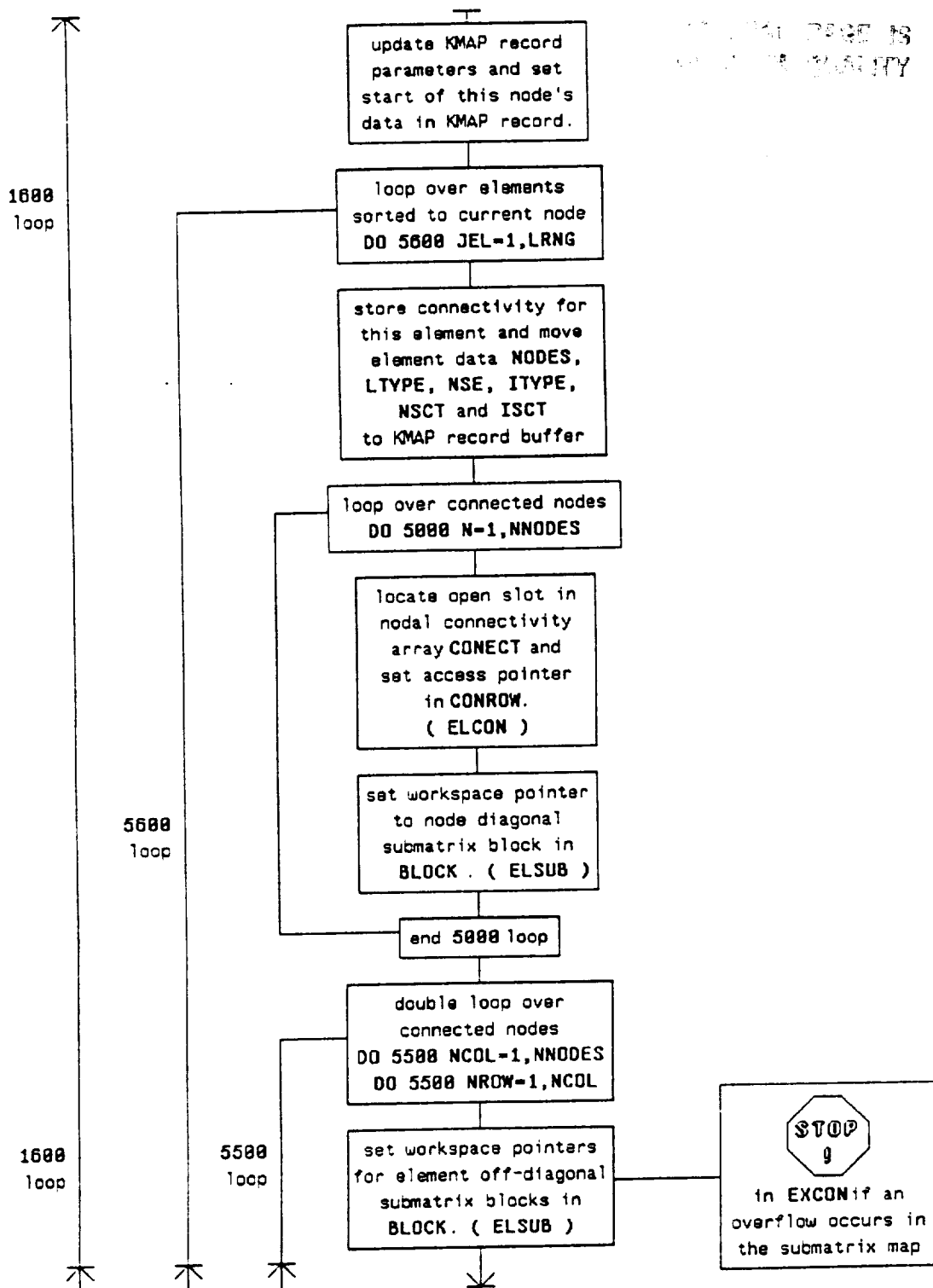


Figure 8 Continued.

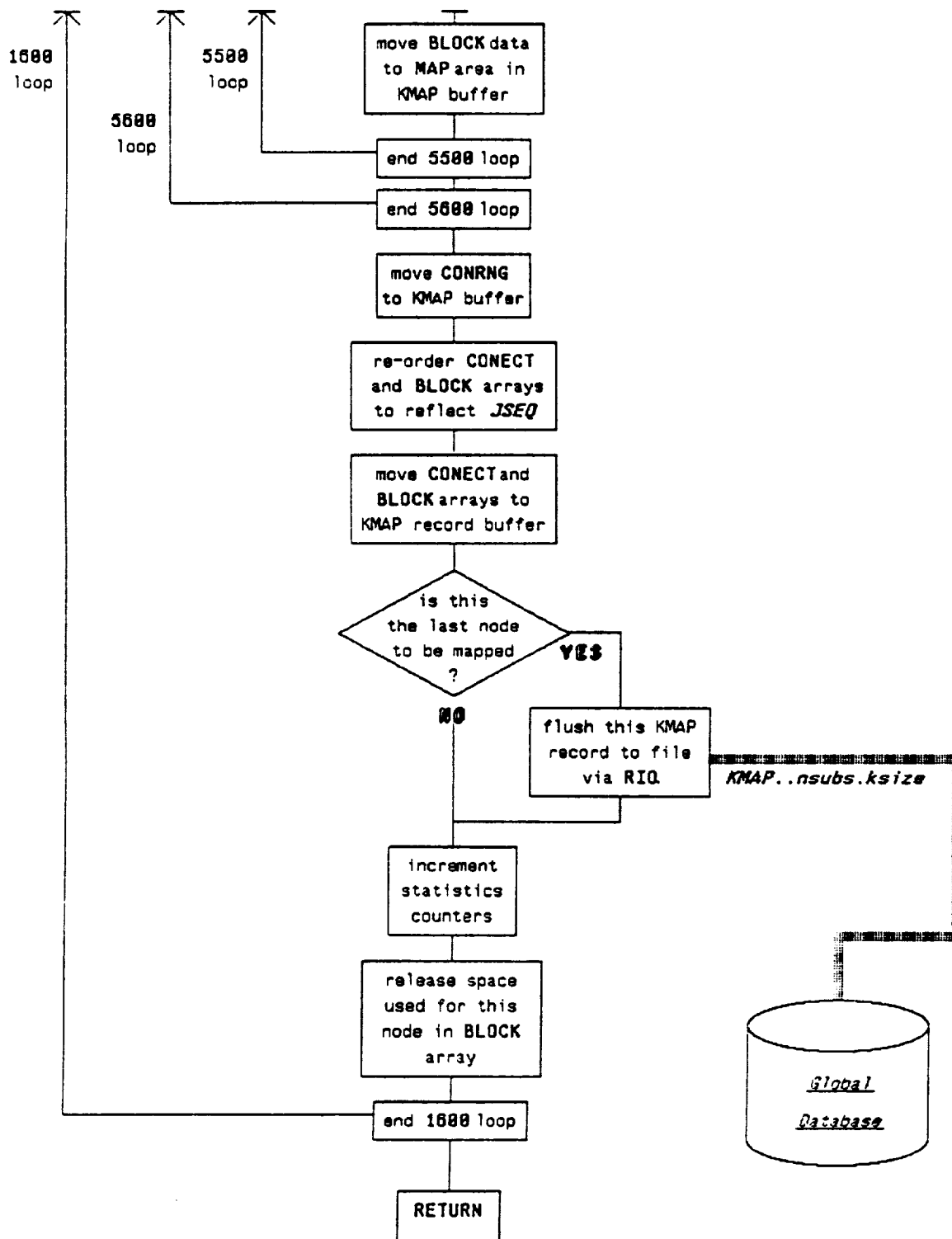


Figure 8 Concluded.



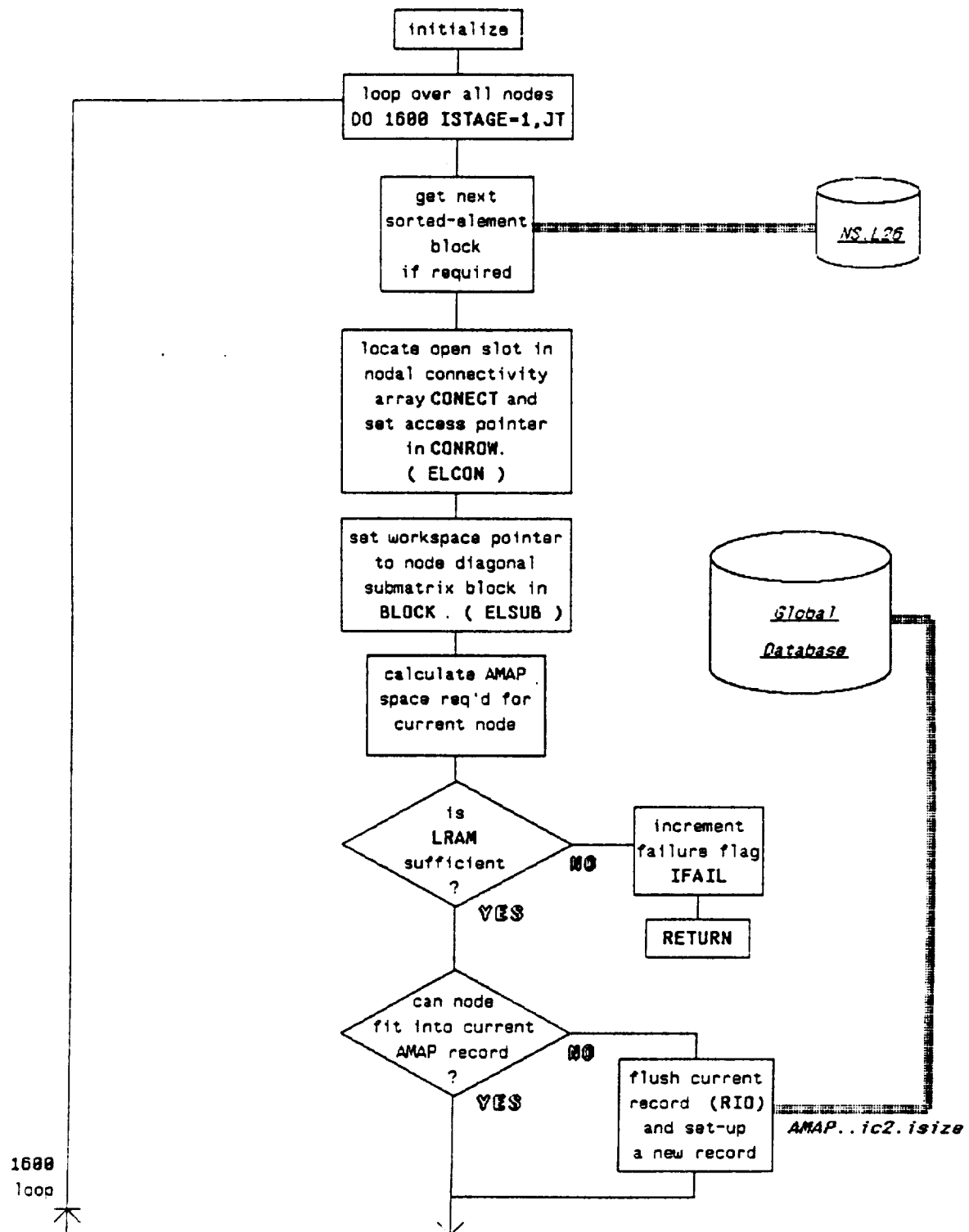


Figure 9 PRECON logic flowchart.

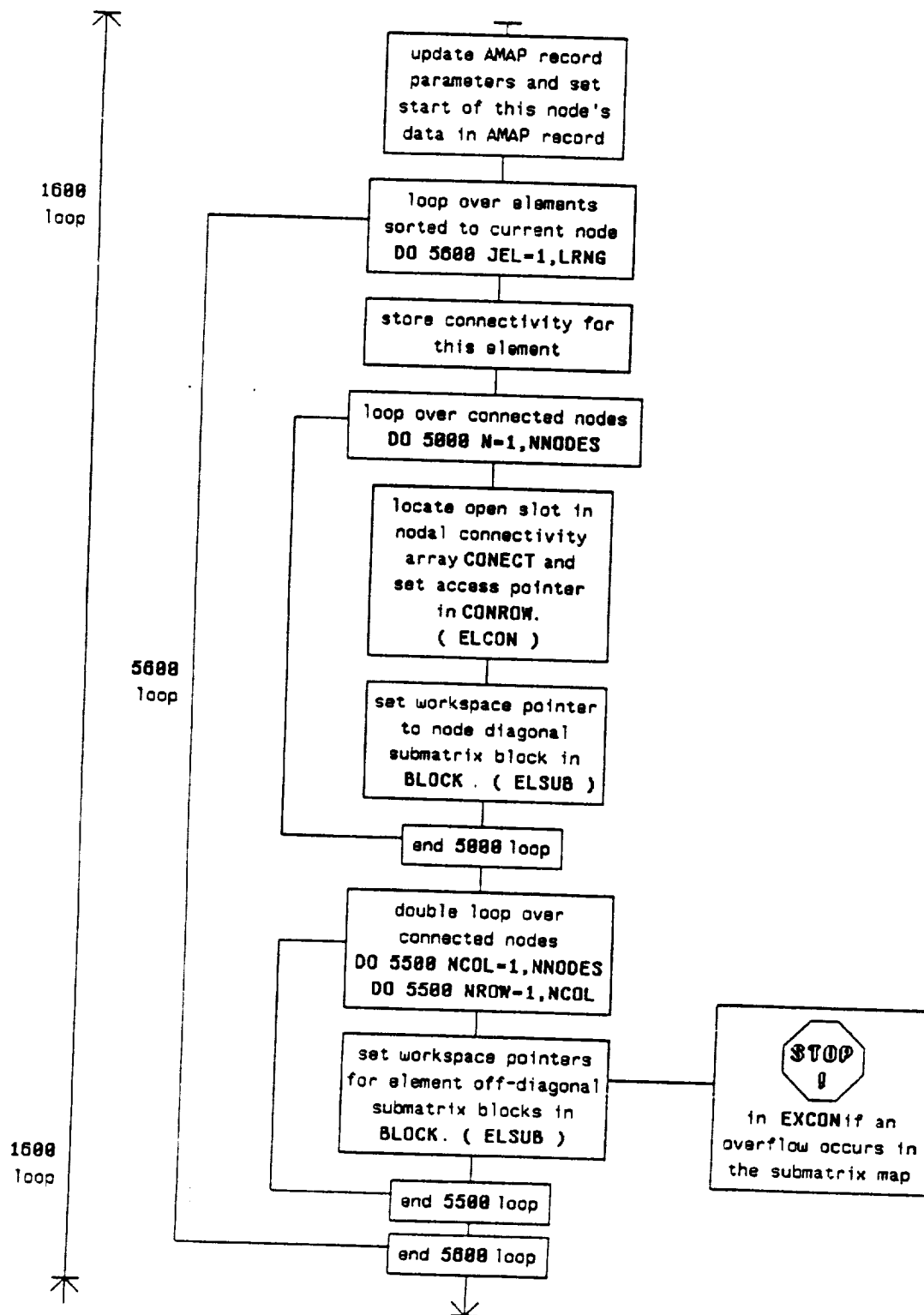


Figure 9 Continued.

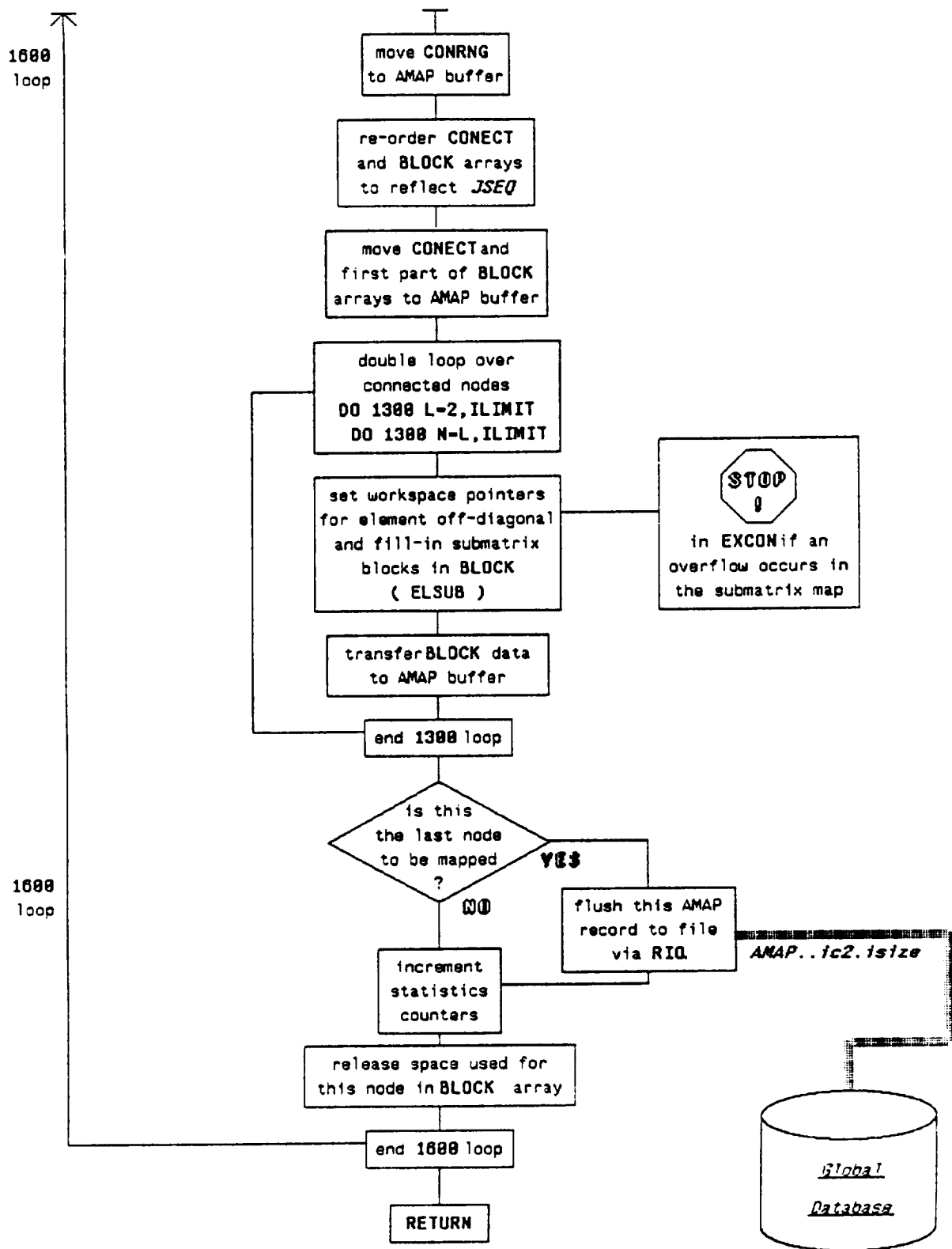


Figure 9 Concluded.

### 2.1.6 PROCESSOR DATA FLOW

Local data management in TOPO is separated into the performance of two functions; the sorting of elements connected to a given node, and the generation of the "MFILES," or matrix maps.

The element data stored while sorting the elements is kept temporarily in LDEF and consists of lowest-referenced node number, element logical type index, number of nodes per element, element physical type and number, section property indices and connected node numbers. Since the looping is over element type code (itype in the DEF.xxxx.itype.nnod datasets), elements with any given lowest-node number are encountered randomly. They are processed as they are encountered and a directory to the records output to file L25 is kept in array ISECT. The number of records written to L25 associated with a particular node is stored in NDUMP.

The size and number of the blocked nodal records on L26 relative to problem parameters determine the adequacy of the core allocation for element sorting calculated in TOPOEX. If an error occurs because too little space has been allocated in TOPOEX for the sorted element data (parameters LMAX, NBLOKS), the element sorting in ELSORT is attempted with a larger value of LMAX. If the element sort has failed three consecutive times, it is reasoned that the problem is simply too large for TOPO to handle given available core space and the routine is aborted.

The generation of the MFILES, KMAP..nsubs.ksize and AMAP..ic2.isize, hinges on the management of the CONROW, CONECT, BLOCK and AVAIL arrays. These arrays are sized based on trial values for ILMAX, MAXCON, MAXSUB, LT4, LT5, LR4 and LR5. These values are updated if the MFILE generation fails. If the map generation fails three times, it is reasoned that the problem is simply too large for TOPO to handle given available core space and the routine is aborted.

### 2.1.7 SUBROUTINE AND VARIABLE NAME GLOSSARY

<i>Subroutine</i>	<i>Description</i>
ELCON	manage the CONECT array
ELSORT	sort elements according to lowest connected node
ELSUB	find next available submatrix location - BLOCK array
EXCON	print error message and abort if BLOCK is overfull
KMAP	construct the KMAP once elements are sorted
PRECON	construct the AMAP once elements are sorted
TOPOEX	main driver routine for construction of maps
TOPOLD	startup routine - resets, number of joints, scratch libraries

<i>Variable</i>	<i>Routine(s)</i>	<i>Description</i>
AVAIL	KMAP,PRECON	AVAIL(k) = .TRUE. if submatrix k is available
BLOCK	KMAP,PRECON	BLOCK(i,j) = submatrix number for CONECT(i,j)
CONECT	KMAP,PRECON	CONECT(i,j) = i-th node connected to CONECT(1,j)
CONRNG	KMAP,PRECON	number of joints attached to current joint (incl joint)
CONROW	KMAP,PRECON	CONROW(i) = column number in CONECT of joint i
IFAIL	ELSORT	failure flag
ILMAX	TOPOEX	max nodal connectivity allowed
IPSA	ELSORT	print flag (print if greater than 0)
ISCT	KMAP	index of section property dataset entry
ISECT	ELSORT	directory of records written to NU6
ITYPE	KMAP	pointer into NS dataset for this element type
JPB	TOPOEX	joints per block for element sort
KA	TOPOEX	workspace
KAC	ELSORT	nodewise element definition data buffer
KFINAL	ELSORT	sorted element data buffer
LB4, LB7	TOPOEX	core allocation pointers during map generation
LDEF	ELSORT	space to hold definition of one element
LLDEF	ELSORT	length of DEF.xxxx.itype.nnod record
LMAX	TOPOEX	max elements per block
LPJ	TOPOEX	approx average elements per joint
LR4	TOPOEX,PRECON	AMAP block size
LR5	TOPOEX,KMAP	KMAP block size
LRECL	TOPOEX,ELSORT	maximum element block size
LRNG	KMAP,PRECON	number of elements attached to current joint
LT1-LT8	TOPOEX	core allocation pointers during map generation
LTYPE	KMAP	logical element type
MAXCON	TOPOEX	max active joints
MAXNN	TOPOEX	max nodes per element
MAXSUB	TOPOEX	max active submatrices
NBLOKS	TOPOEX,ELSORT	number of blocks joints divided into for element sort
NDUMP	ELSORT	number of dumps per block to NU6 during coarse sort
NELS	ELSORT	number of elements in definition record
NETOT	TOPOEX	padded estimate of total number of elements
NNODES	KMAP,ELSORT	number of nodes attached to this element
NSCT	KMAP	N4 of section property dataset name for this type
NSE	KMAP	element number within this type
NU6	ELSORT	scratch library used during coarse element sort
NU7	ELSORT	scratch library for final element sort

### 2.1.8 USAGE GUIDELINES AND EXAMPLES

An example of the use of the KMAP data structure is contained in the section on the K processor.

THIS PAGE LEFT BLANK INTENTIONALLY.

## 2.2 Processor K

### 2.2.1 GENERAL DESCRIPTION

Processor K is responsible for expansion (in some cases) and assembly of element stiffness matrices into the sparse format system stiffness matrix.

K is really a fairly simple processor. Most of the bookkeeping done in K is related to the distributed storage scheme used for element data. As such, element directories, name tables, and much extraneous EFIL data need to be stored merely to facilitate the acquisition of element nodal submatrix data for assembly.

### 2.2.2 PROCESSOR SYNTAX

This processor follows Testbed command syntax and data management conventions as described in Reference 2.

#### 2.2.2.1 Processor Resets

<i>Argument</i>	<i>Default</i>	<i>Meaning</i>
LREC	2240	output matrix record length
SA	0	Diagnostic print flag – core allocation
BLIB	1	Input library number for *.BTAB and DEF.*
ELIB	1	Input library number for *.EFIL.*
HLIB	1	Input library number for KMAP..nsubs.ksize
OUTLIB	1	Output (matrix) library
SPDP	1 or 2	Single- or double-precision output matrix
NAME	K	First field of output matrix dataset name

### 2.2.3 SUBPROCESSORS AND COMMANDS

Not applicable.

## 2.2.4 PROCESSOR DATA INTERFACE

### 2.2.4.1 Processor Input Datasets

- JDF1.BTAB.1.8
- NS
- ELTS.NAME
- \*.EFIL.\*
- KMAP..nsubs.ksize

### 2.2.4.2 Processor Output Datasets

- K.SPAR.jdf2 (or name.SPAR.jdf2)

## 2.2.5 PROCESSOR LOGIC FLOW

Figures 10 through 14 contain logic flowcharts for key subroutines in K: the main subroutine, subroutine KSMLD which handles RESET processing and some core workspace allocation, subroutine ASKGO, subroutine ASKEX which directs matrix assembly, and subroutine SPTRN\* which transforms and assembles element stiffness matrices into the global system stiffness matrix.

It is important to remember that K functions as much more than a matrix assembler. In particular, K is responsible for the expansion of the SPAR "intrinsic" element stiffness matrices for beam, plate and solid elements in to their full matrix forms. K also transforms these matrices from element-local coordinate frames to the global system frames using the transformation information stored in the EFIL. Lower-level subroutines such as ADDH, TRIL and TRIL3 perform the expansion of the intrinsic stiffnesses. As such, these are really element-specific routines embedded in the matrix assembler!

The \*TRN\* (*e.g.*, SPTRN6, DPTRN3) subroutines are the actual assembly subroutines. A generic example of this type of routine is the flowcharted SPTRN6 subroutine. These subroutines take element stiffness blocks and transfer appropriate, transformed nodal submatrices into the S workspace for direct assembly into the output matrix buffer in subroutine ASKEX (or double precision routine DASKEX).

All assembly is performed on an element-by-element basis for a particular node, *i.e.*, all elements referencing a certain node and higher nodes (in the sense of the elimination sequence) are read-in for assembly to that certain node's portion of the system matrix block. Hence, the outer loop over number of nodes and the inner loop over the attached elements, as directed by KMAP..nsubs.ksize.



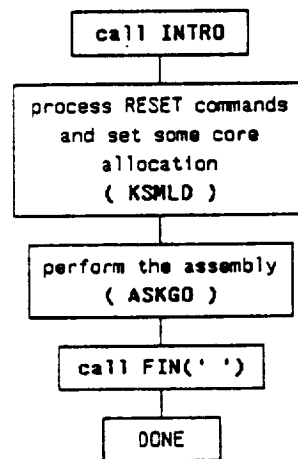


Figure 10 K main program logic flowchart.

ORIGINAL PAGE IS  
OF POOR QUALITY

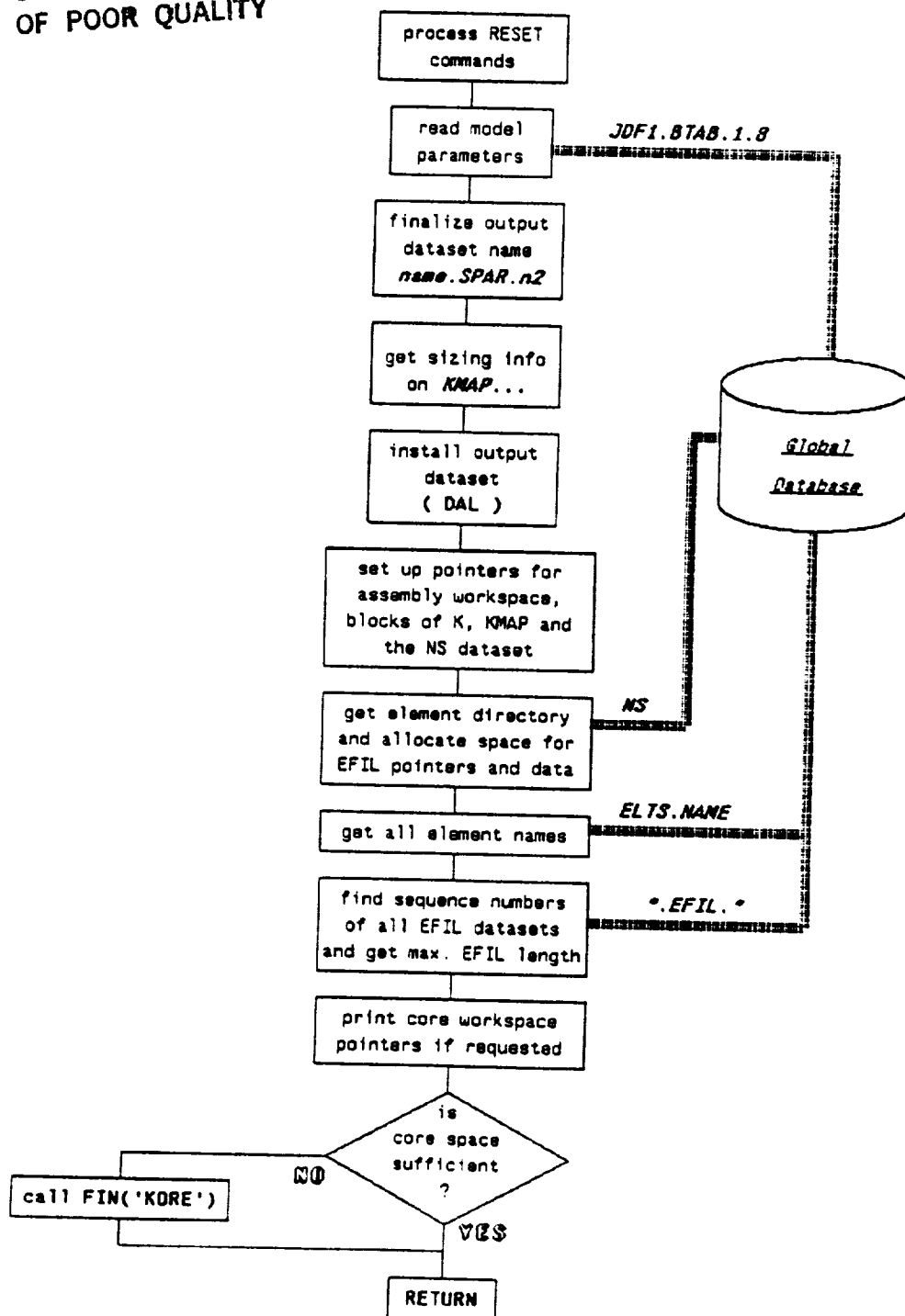


Figure 11 KSMLD logic flowchart.

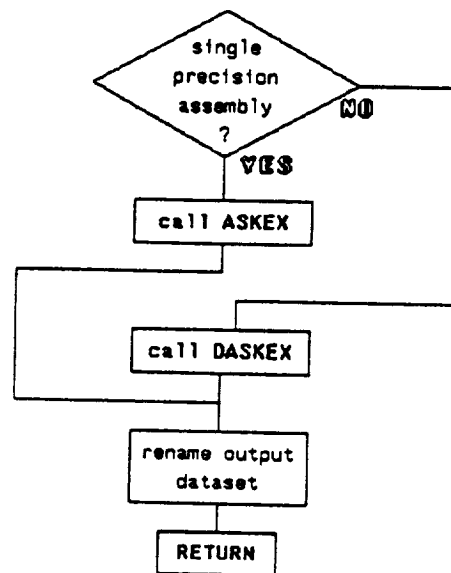


Figure 12 ASKGO logic flowchart.

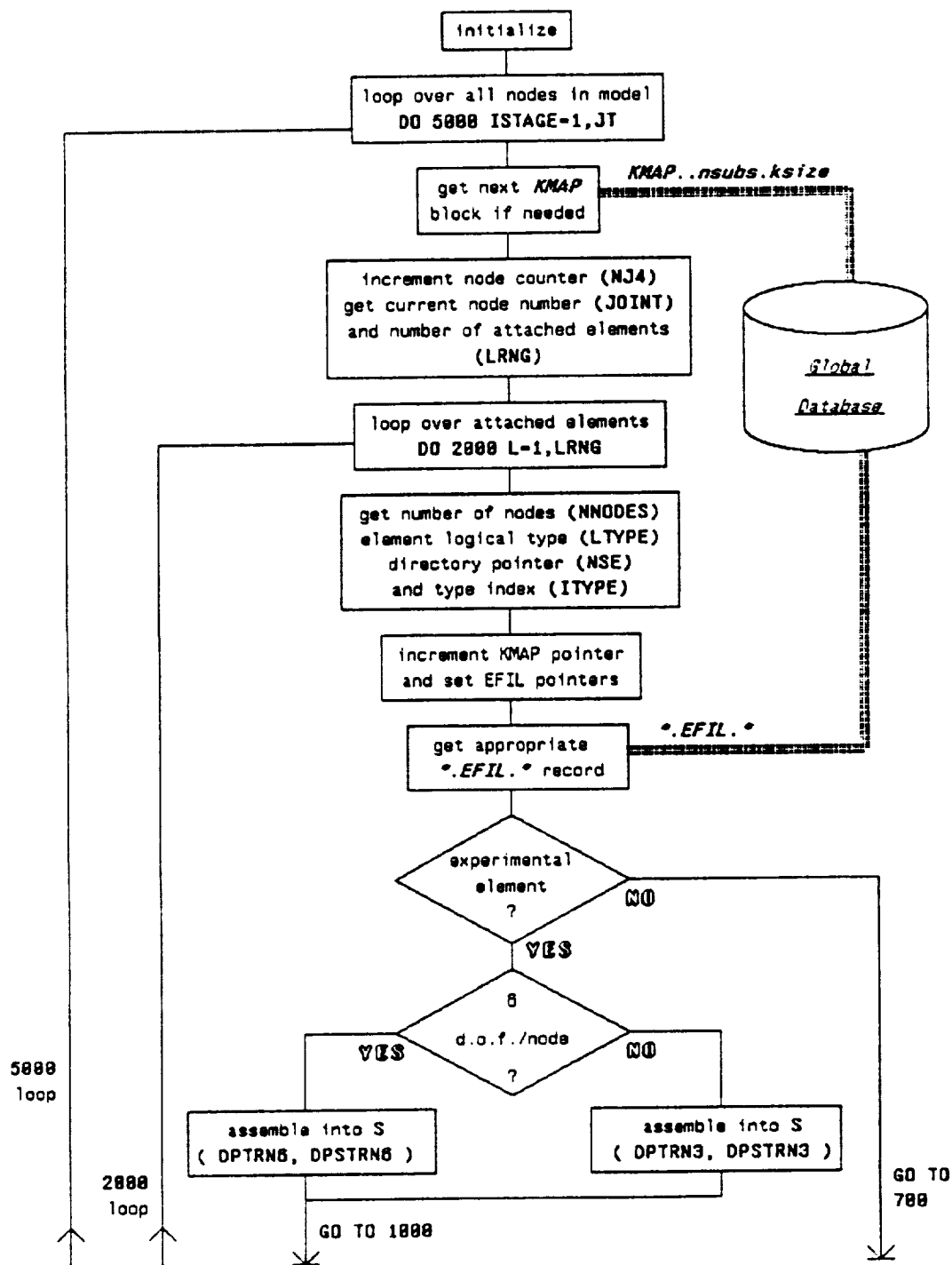


Figure 13 ASKEX (DASKEX) logic flowchart.

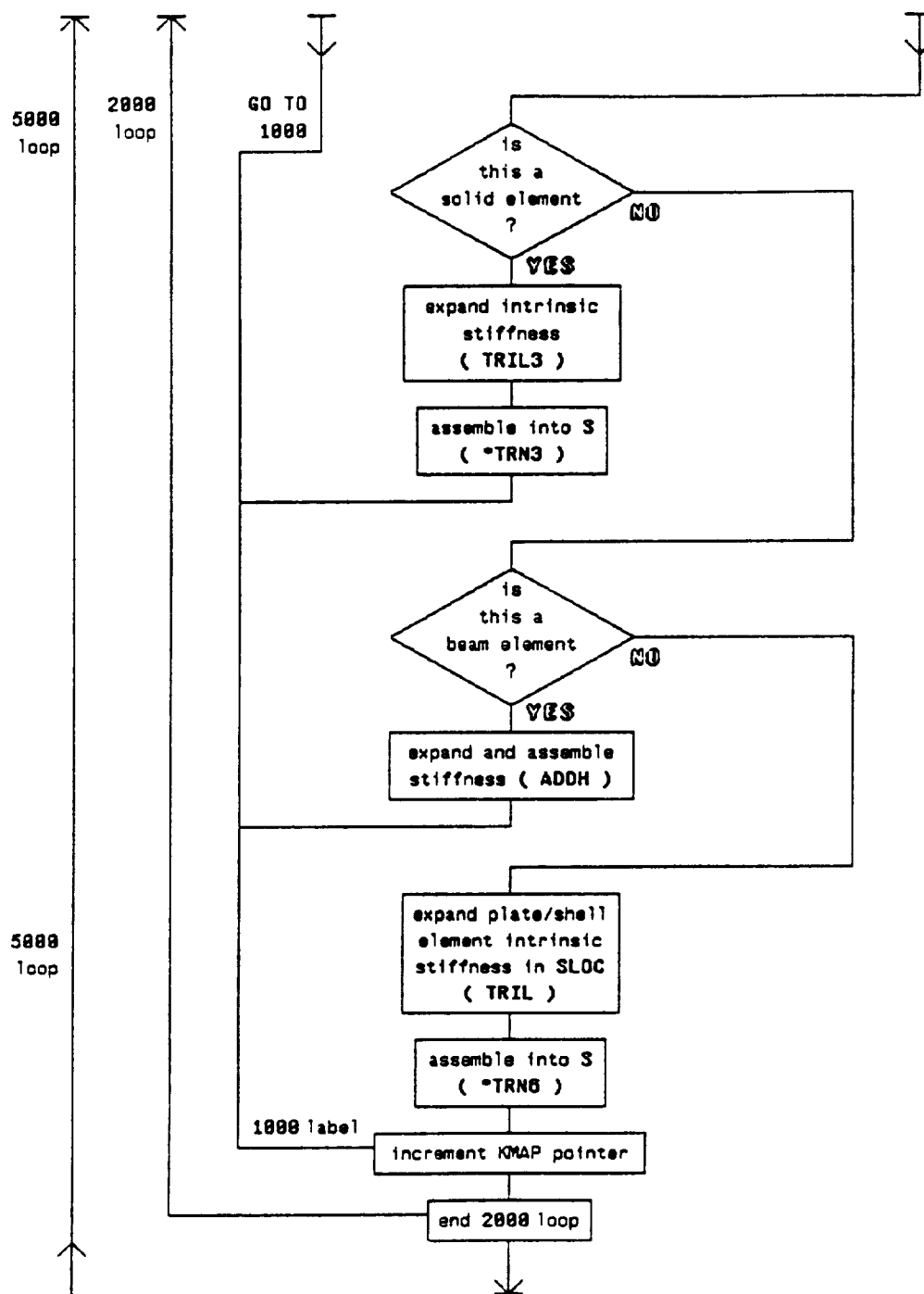
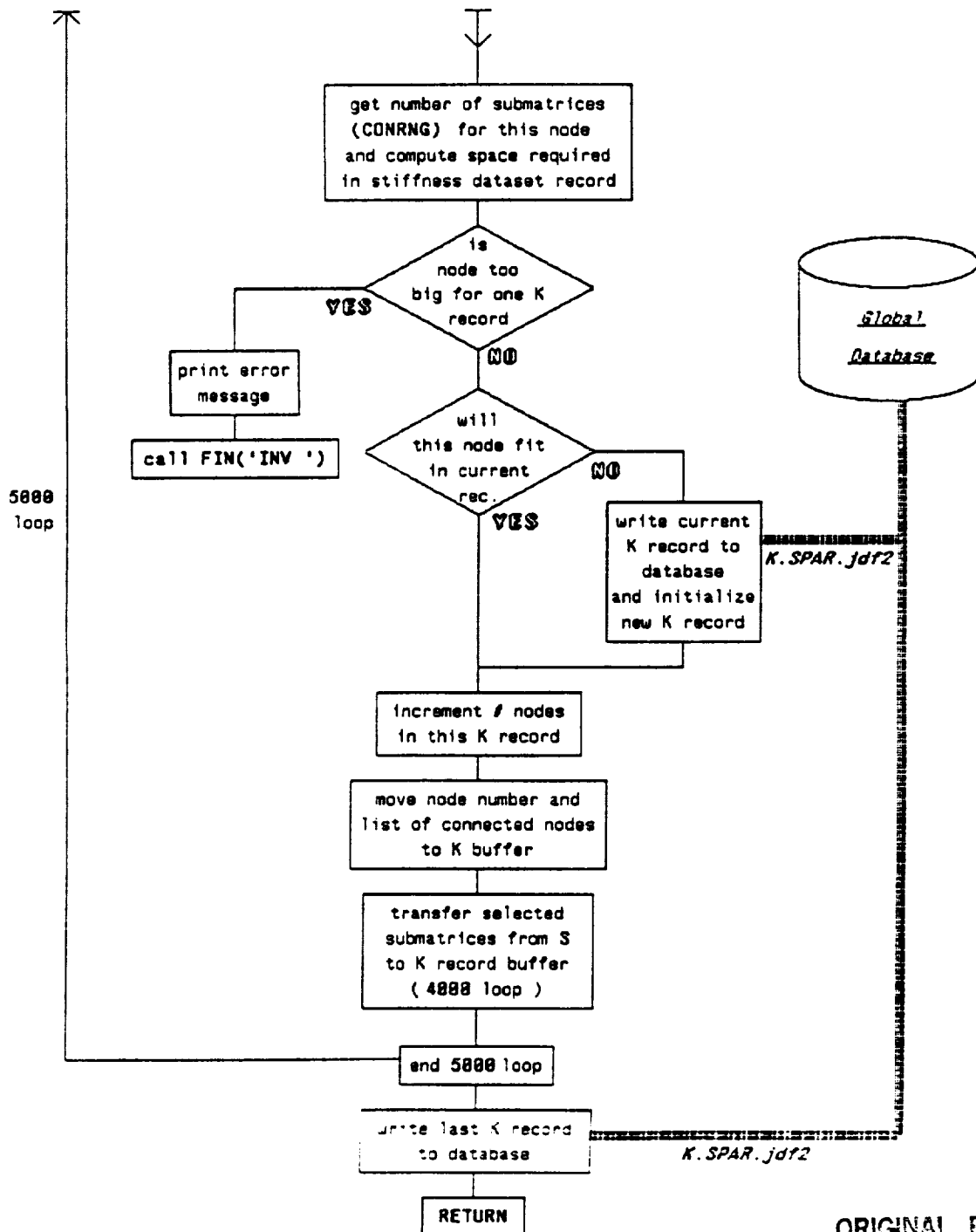


Figure 13 Continued.



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 13 Concluded.

ORIGINAL PAGE IS  
OF POOR QUALITY

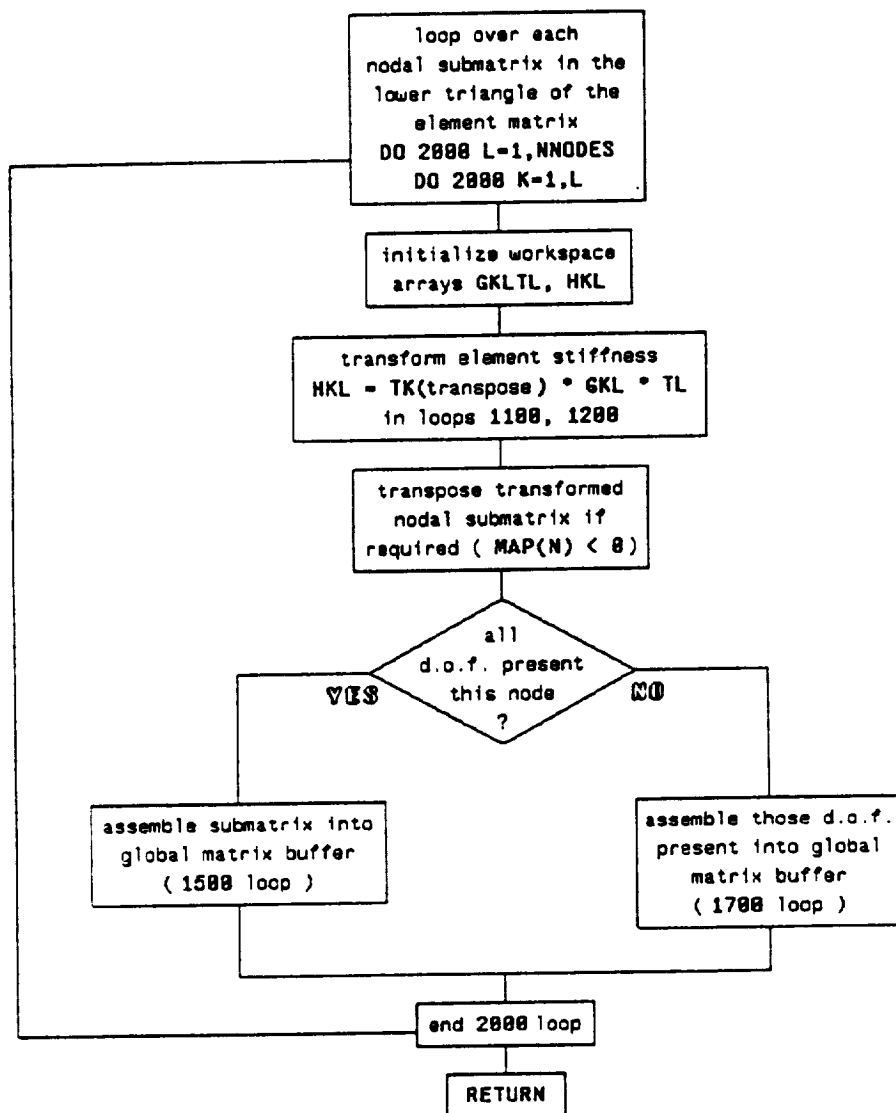


Figure 14 TRN3 (TRN6) logic flowchart.

### 2.2.6 PROCESSOR DATA FLOW

Workspace storage is allocated for the S array, which holds the expanded, transformed, nodal submatrix blocks for assembly. The S array is sized to hold KSIZE submatrix blocks dimensioned at NDF by NDF (NDF = number of degrees-of-freedom per node). KSIZE is defined in the KMAP..nsubs.ksize. Workspace is also allocated for one block of the output system matrix, one block (record) of the KMAP..nsubs.ksize, the entire contents of the NS dataset, a vector of dataset sequence numbers for all \*.EFIL.\* datasets, the contents of the ELTS.NAME dataset, and the longest EFIL record present in the model. All of these allocations are made in subroutine KSMLD.

Additional EFIL pointers are used in ASKEX (DASKEX) to assist in decoding the rather cryptic EFIL internal record structure.

### 2.2.7 SUBROUTINE AND VARIABLE NAME GLOSSARY

<i>Subroutine</i>	<i>Description</i>
ADDH	assemble beam elements
ASKGO	switches on single/double to call ASKEX or DASKEX
ASKEX(DASKEX)	main driver for assembly
KSMLD	startup resets, dataset access, preliminary core allocation
SPTRN3	assemble 3 d.o.f. per node elements with transformation
SPTRN6	assemble 6 d.o.f. per node elements with transformation
DPTRN3	double precision output version of SPTRN3
DPTRN6	double precision output version of SPTRN6
DPSTRN3	double precision input/output version of SPTRN3
DPSTRN6	double precision input/output version of SPTRN6
TRIL	expand intrinsic element stiffness (shells)
TRIL3	expand intrinsic element stiffness (solids)



<i>Variable</i>	<i>Routine(s)</i>	<i>Description</i>
CONRNG	ASKEX	number of submatrices attached to current node
GKLTL	*TRN*	transformation workspace = Klocal * Transform
HKL	*TRN*	transformation workspace = Kglobal
ITYPE	ASKEX	pointer into NS dataset for this element type
LRNG	ASKEX	number of elements attached to current node
LTYPE	ASKEX	logical element type
MAP	*TRN*	submatrix location vector for assembly
NJ4	ASKEX	joint counter within current block of KMAP
NNODES	ASKEX	number of nodes attached to current element
NSE	ASKEX	element number within this type
S	ASKEX	submatrix assembly workspace
SLOC	ASKEX	local workspace to hold expanded element stiffness

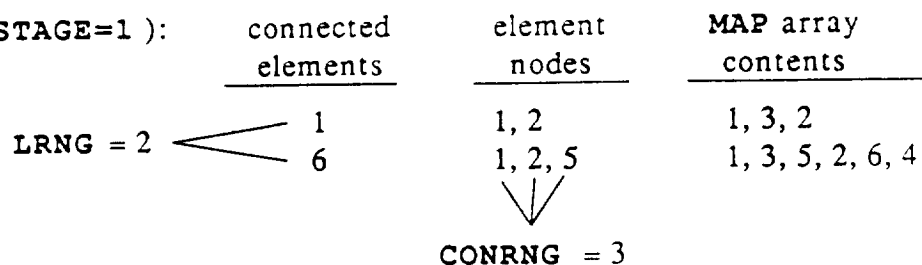
## 2.2.8 USAGE GUIDELINES AND EXAMPLES

An example of the use of the information contained in the KMAP..nsubs.ksize dataset to assemble the stiffness matrix of the example model of §1.3 is presented in figure 15. The assembly of the first two nodal-block rows of the upper triangle of the system matrix are shown in detail. The interested reader will find the logic flowcharts (see fig. 13) for subroutines ASKEX and DASKEX to be helpful in understanding this example. The steps in the illustrated assembly process are as follows:

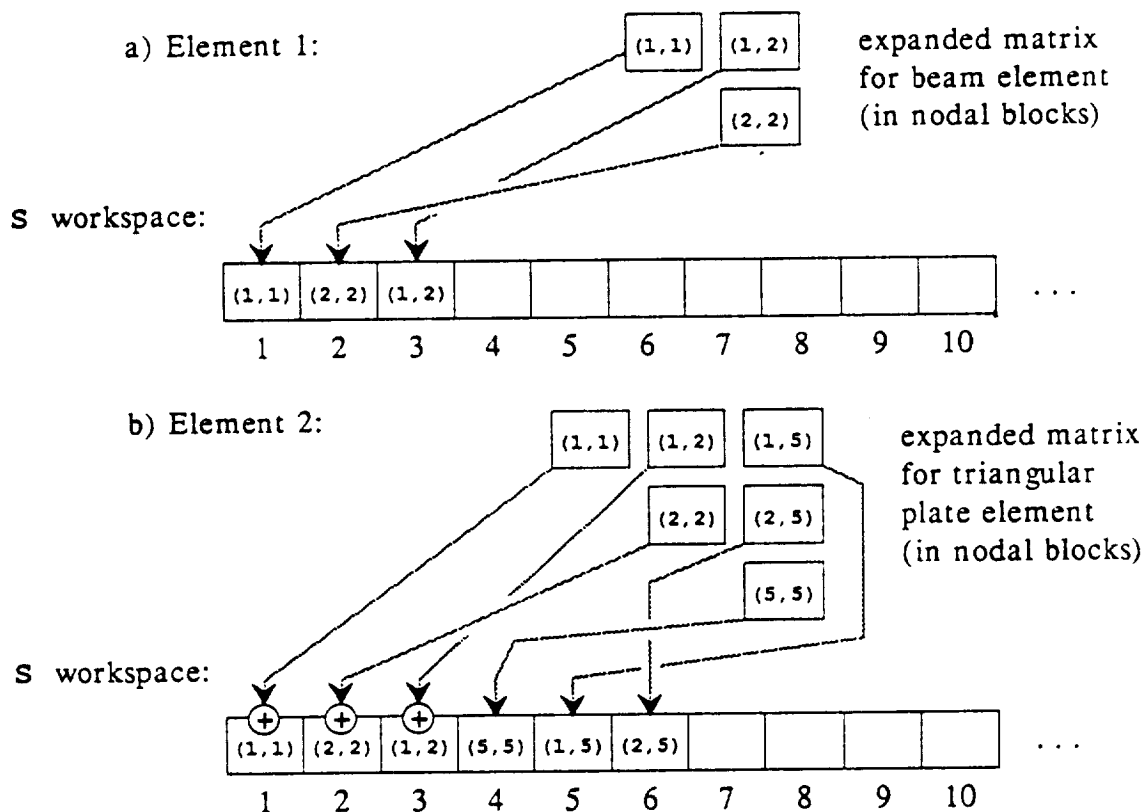
- A) Loop over all nodes in the model. For the first node:
  - 1) Assemble, the nodal-block submatrices for each of the LRNG elements whose lowest-numbered node (in the sense of the elimination sequence) is the current node. The assembly is accomplished by accumulating nodal-block submatrices from all contributing elements into appropriate slots in the S workspace. The allocation of these S workspace slots is done in TOPO and communicated by the MAP array in the KMAP..nsubs.ksize data structure. Note that negative indices in the MAP array indicate that the transpose of the element nodal-block submatrix is to be assembled into S.
    - a) The first element is a (two-node) beam element. Therefore, three nodal-block submatrices form the upper triangle of the element matrix. These three submatrices are moved into slots 1, 2 and 3 in the S workspace. Since this is the first element, the nodal submatrices do not need to be accumulated into the previous contents of the S workspace.

- b) The second element is a triangular plate element having six nodal blocks in the upper triangle of its element matrix. These submatrices are accumulated into slots 1-6 of the S workspace.
- 2) After accumulation of the nodal submatrices from the second element's matrix into S, the slots corresponding to the first nodal-block row of the system matrix (1, 3 and 5) are completely assembled and can be moved to the system matrix record buffer. Once transferred, these blocks in the S workspace are no longer needed and are free to be used for accumulation of other submatrices.

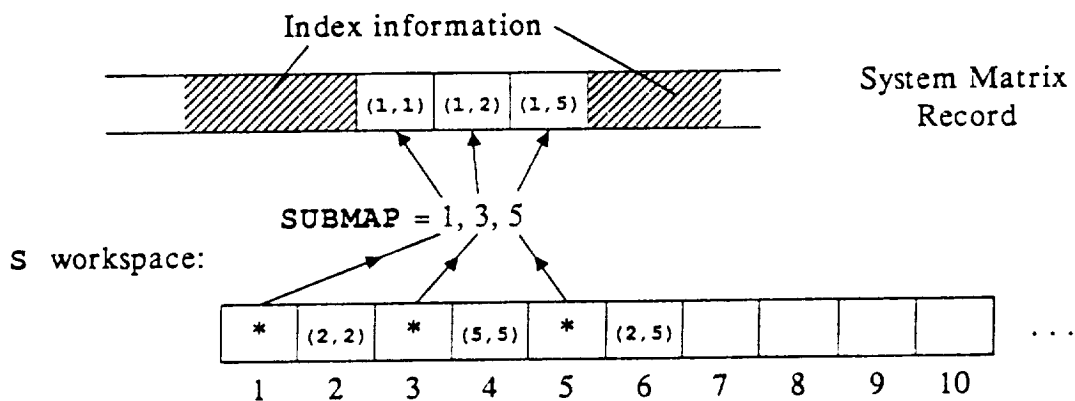
The above process is repeated for each of the six nodes in the example model.

A) for node 1 ( **ISTAGE=1** ):

## 1) Assembly of Element Stiffness Matrices :



## 2) Transfer to System Matrix Record Buffer:



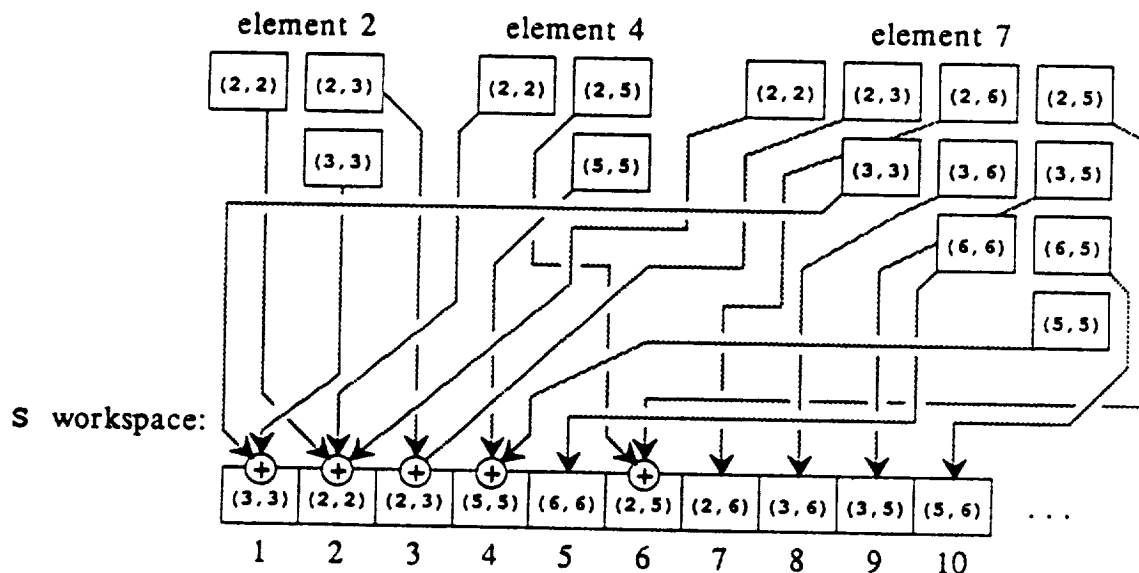
\* - Block is now free for accumulation of other nodal submatrices

Figure 15 Example of the system matrix assembly process.

B) for node 2 ( Istage=2 ):

	connected elements	element nodes	MAP array contents
LRNG = 3	2	2, 3	2, 3, 1
	4	2, 5	2, 6, 4
	7	2, 3, 6, 5	2, 3, 7, 6, 1, 8, 9, 5, -10, 4
		CONRNG = 4	

1) Assembly of Element Matrices:



2) Transfer to System Matrix Record Buffer:

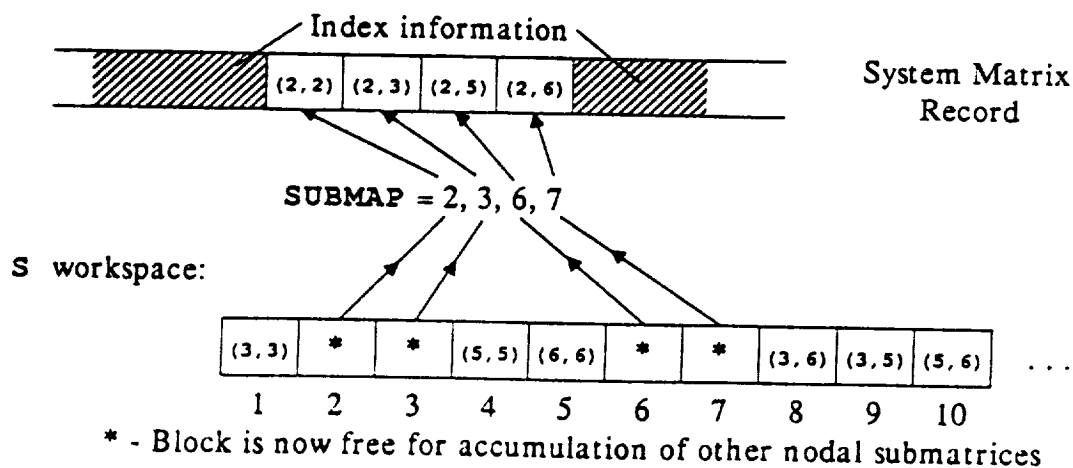


Figure 15 Concluded.

## 2.3 Processor INV

### 2.3.1 GENERAL DESCRIPTION

Diagonal-scale factoring of a system matrix.

$$\mathbf{LDL}^T = \mathbf{A}$$

- $\mathbf{A}$  is a sparse system matrix
- $\mathbf{L}$  and  $\mathbf{D}$  are stored as the inverted matrix

The operation of INV reflects a conventional diagonal-scale factoring process. The only specialization is for the treatment of prescribed degrees-of-freedom where the factoring process is stopped in a column for the row where a prescribed d.o.f. is encountered and for succeeding rows. Unfactored coefficients are kept in these untouched positions, and the forward-reduction process in SSOL is structured to ensure the proper treatment of linear internal forces associated with these prescribed d.o.f. Thus, the INV and SSOL processors exhibit interdependence through both the factored matrix data structure and the specialized treatment of applied displacements.

When interpreting the factored matrix database data structure one needs to be aware that the nonzero d.o.f. map in the INV record for a particular node actually contains pointers to those d.o.f. not constrained on the *START* card (in TAB) that are also not constrained in the selected CON data. Rows corresponding to constrained d.o.f. are not present in the factored matrix. As such, knowledge of the basic problem parameters stored in JDF1.BTAB.1.8 is required to decode fully the factored matrix data structure.

### 2.3.2 PROCESSOR SYNTAX

This processor follows Testbed command syntax and data management conventions as described in Reference 2.

### 2.3.2.1 Processor Resets

<i>Argument</i>	<i>Default</i>	<i>Meaning</i>
K	K	First field of input matrix dataset name (will also be second field of output factored matrix dataset name)
DZERO	1.0E-05	Zero test for diagonal terms
SPDP	1	Single (1) or Double (2) precision factored matrix
CON	1	ncon in CON..ncon dataset
KLIB	1	Library containing <b>A</b> matrix
KILIB	1	Destination library for factored matrix
NJMAX	50	Max. number of nodal row blocks in one record of the factored matrix
LRA	3584	Output (factored matrix) dataset record length
ILIB	(KLIB)	Library containing AMAP..ic2.isize

### 2.3.3 SUBPROCESSORS AND COMMANDS

Not applicable.

### 2.3.4 PROCESSOR DATA INTERFACE

#### 2.3.4.1 Processor Input Datasets

- JDF1.BTAB.1.8
- CON..ncon
- AMAP..ic2.isize
- K.SPAR.jdf2 (name.SPAR.jdf2)

#### 2.3.4.2 Processor Output Datasets

- INV.K.ncon (INV.name.ncon)

### 2.3.5 PROCESSOR LOGIC FLOW

Figures 16 through 20 contain logic flowcharts for the major INV subroutines including the main subroutine, subroutine AFLD, subroutine AFG0, subroutine AFEX (double precision version DPAFEX) which directs the factoring, and subroutine RED (double precision version REDDP) which actually performs the elimination calculations. Logic for single or double precision factoring of matrices is separated by subroutine and based on the precision of the input matrix. If double precision factors are requested, the input matrix *must* also be in double precision. If single precision factors are requested (this is the default) and the input matrix is double precision, the factors are truncated in REDDP before being written out in DPAFEX.

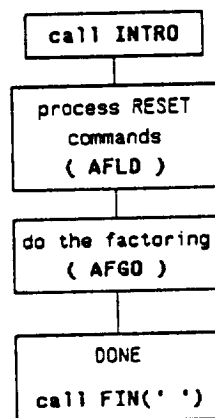


Figure 16 INV main program logic flowchart.

ORIGINAL PAGE IS  
OF POOR QUALITY

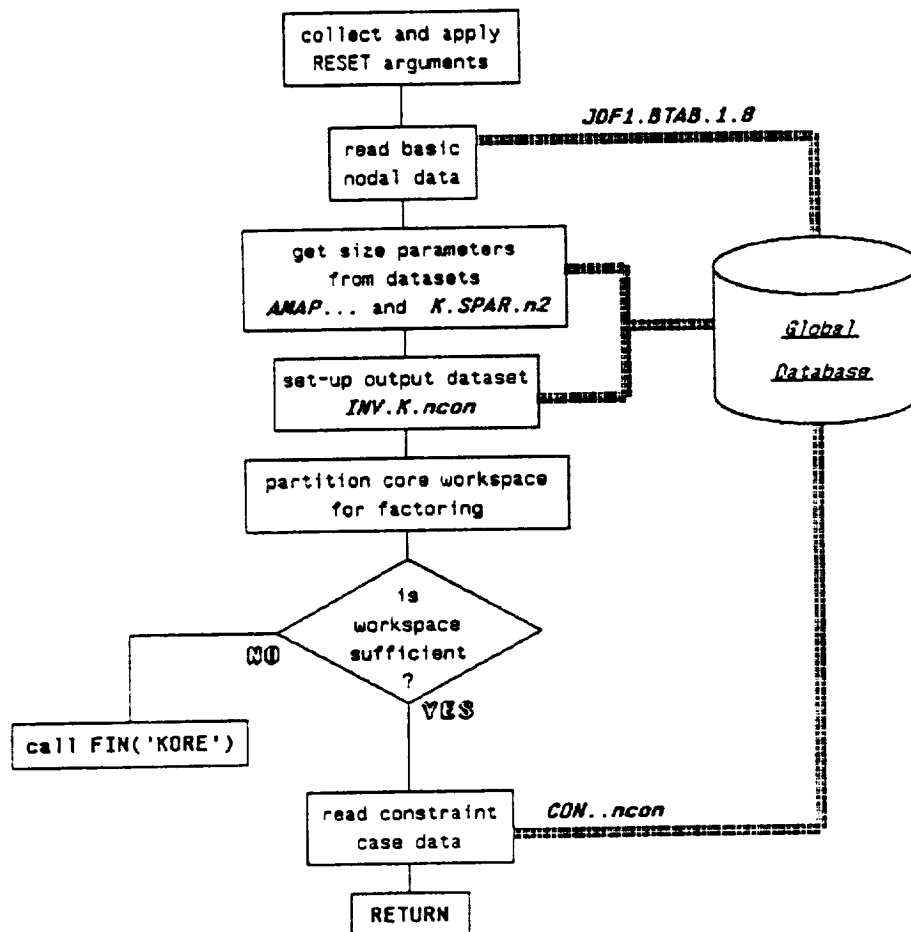


Figure 17 AFLD logic flowchart.



ORIGINAL PAGE IS  
OF POOR QUALITY

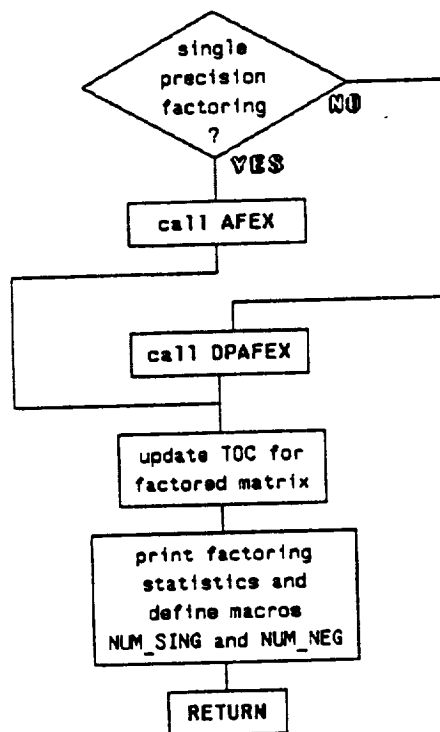


Figure 18 AFGO logic flowchart.

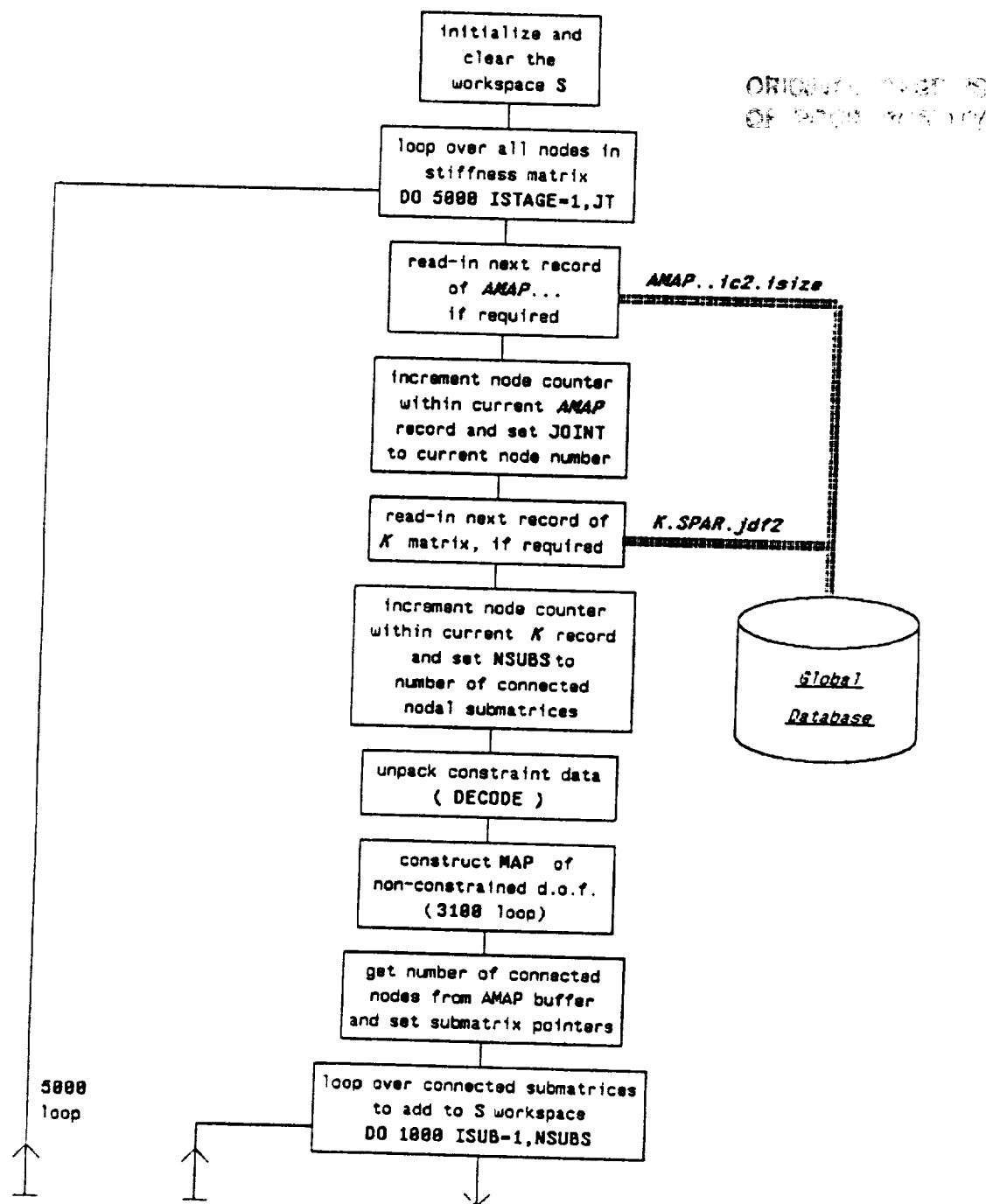


Figure 19 AFEX (DPAFEX) logic flowchart.

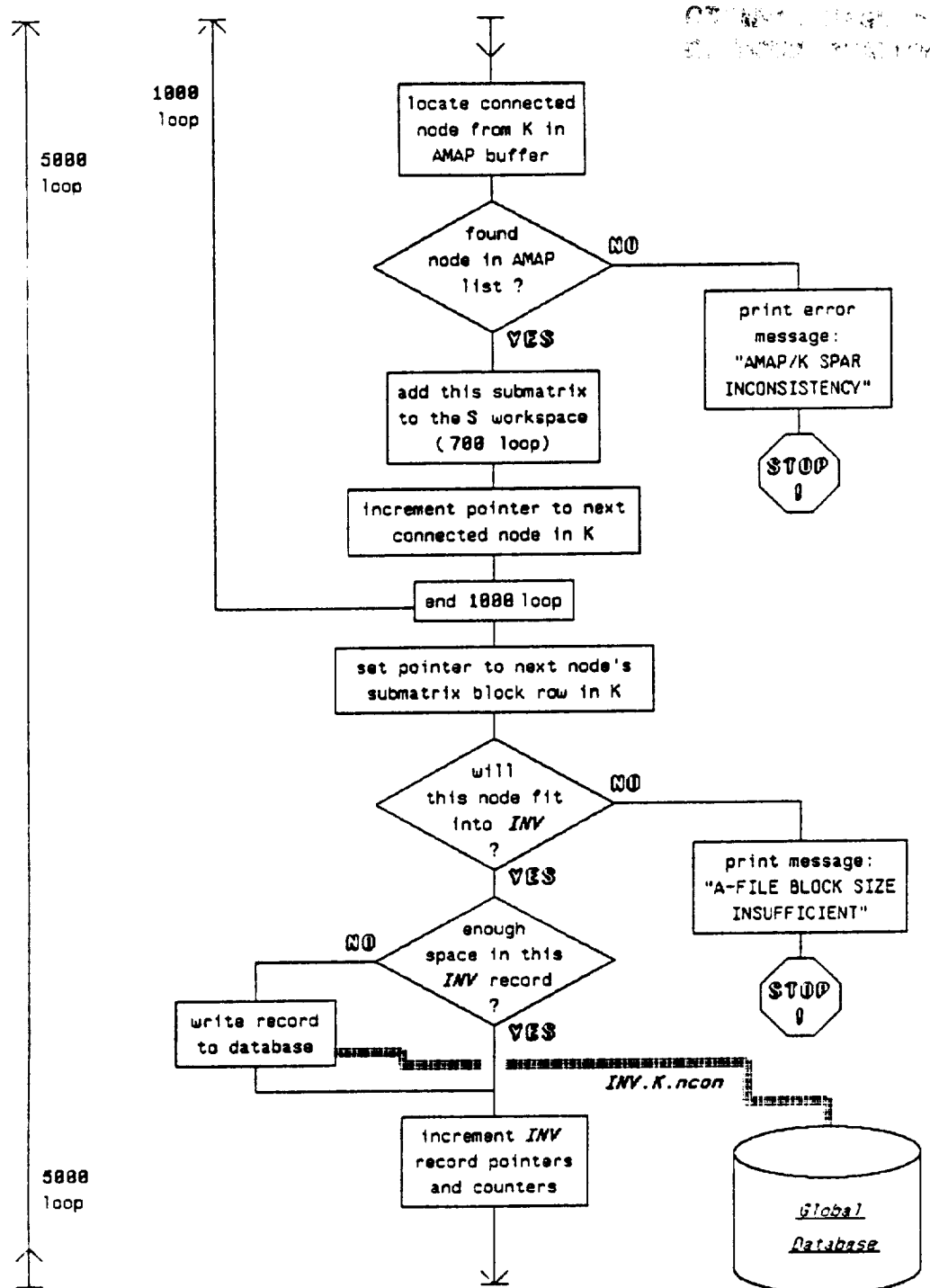
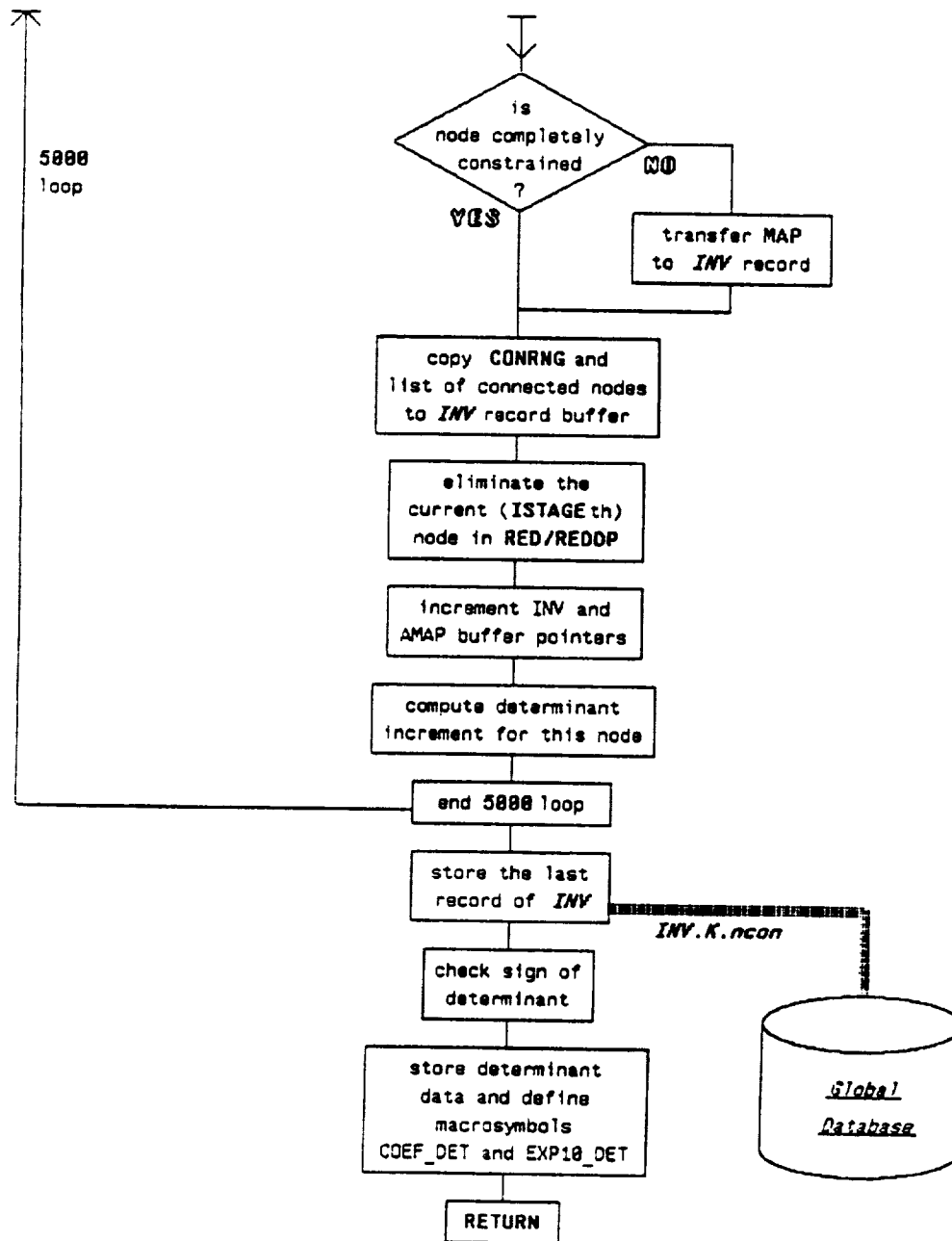


Figure 19 Continued.



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 19 Concluded.

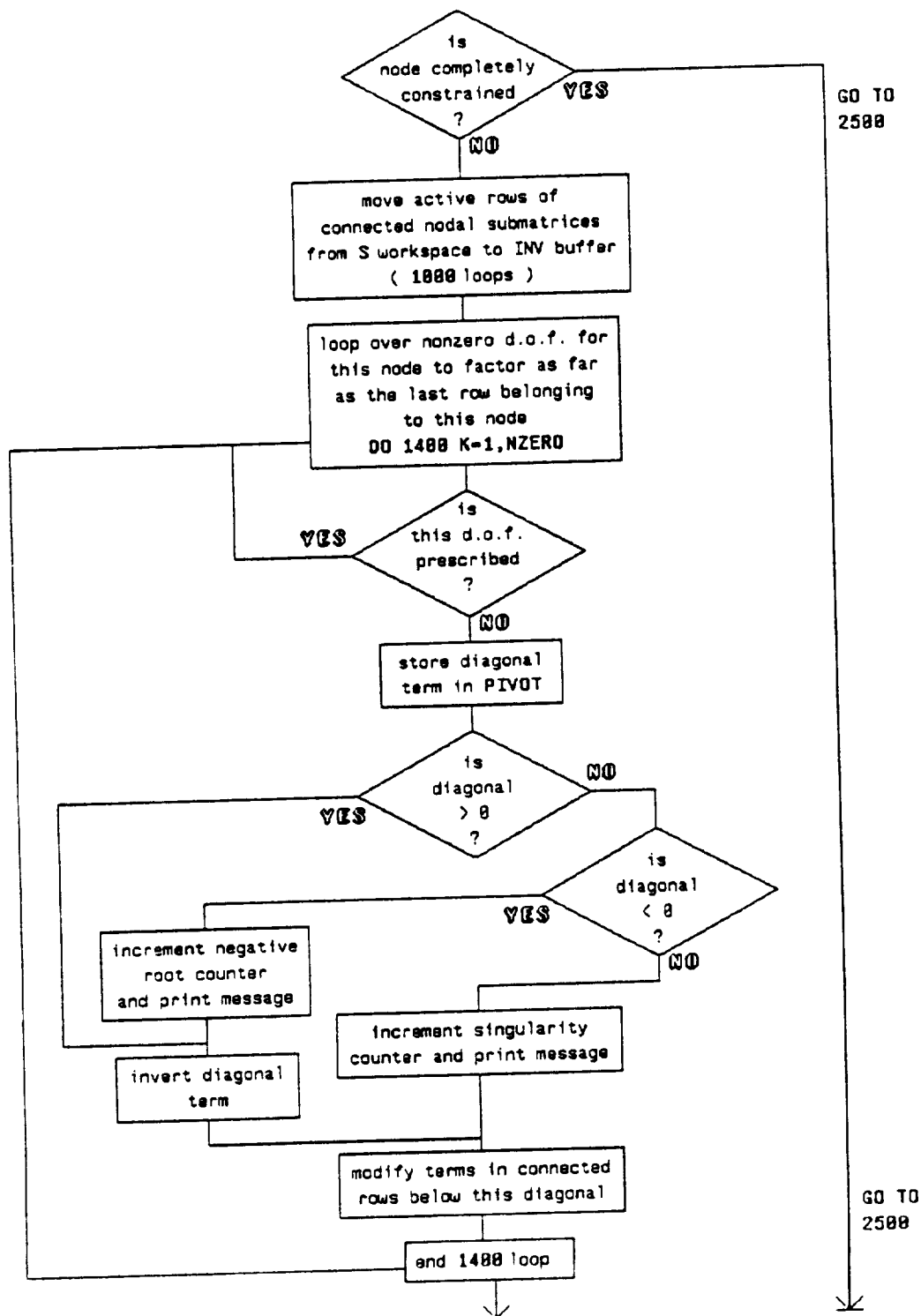
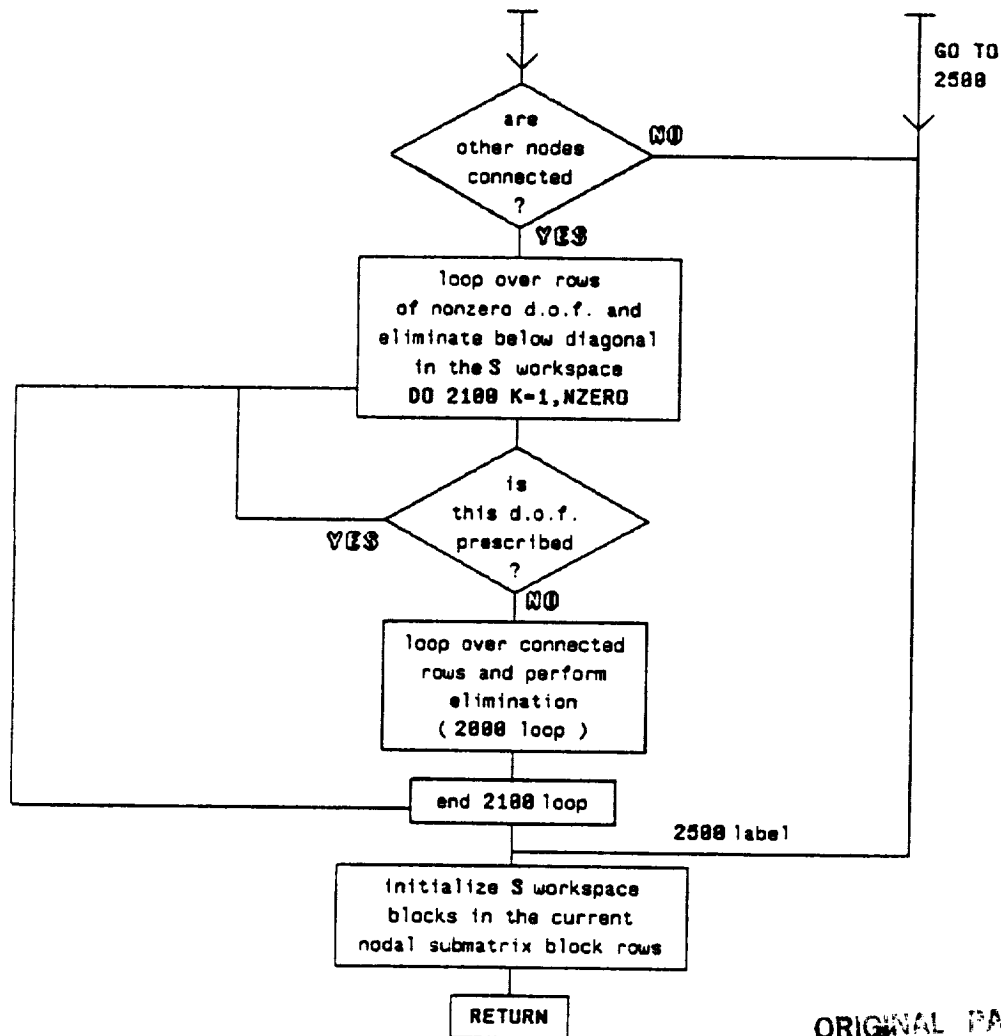
ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 20 RED (REDDP) logic flowchart.



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 20 Concluded.

### 2.3.6 PROCESSOR DATA FLOW

Core workspace is allocated in subroutine AFLD. Workspace for the factoring operation (referred to as the S workspace in the code) is sized to hold isize nodal submatrices as determined from the AMAP generation via the name of the AMAP..ic2.isize dataset. Additional space is allocated for the entire CON..ncon dataset and one record each of the K.SPAP.jdf2, AMAP..ic2.isize and INV.K.ncon datasets. All of the difficult details of managing submatrix blocks in core have been relegated to the topology analysis in TOPO and, by the time INV is invoked, are completely coded into the AMAP data structure.

### 2.3.7 SUBROUTINE AND VARIABLE NAME GLOSSARY

<i>Subroutine</i>	<i>Description</i>
AFEX(DPAFEX)	factorization driver
AFGO	switches on single/double to call AFEX or DPAFEX
AFLD	startup resets, dataset access, core allocation, etc.
DECODE	decodes packed CON..ncon entries
RED(REDDP)	reduction for current joint

<i>Variable</i>	<i>Routine(s)</i>	<i>Description</i>
CONRNG	AFEX	number connected submatrices in factored matrix
MAP	AFEX	map of nonzero dof for current node
JOINT	AFEX	node counter for factoring
NSUBS	AFEX	number connected submatrices in unfactored matrix
NZERO	AFEX, RED	number nonzero dof for current node
PIVOTS	RED	diagonal terms for nonzero dof at current joint
S	AFEX, RED	factorization work area

### 2.3.8 USAGE GUIDELINES AND EXAMPLES

None.

THIS PAGE LEFT BLANK INTENTIONALLY.



## 2.4 Processor SSOL

### 2.4.1 GENERAL DESCRIPTION

- Calculate solutions  $\mathbf{x}$  to

$$\mathbf{Ax} = \mathbf{f}$$

- $\mathbf{A}$  is a sparse system matrix that has been factored by INV
- $\mathbf{f}$  is the aggregate of applied nodal forces and element distributed loads
- $\mathbf{x}$  is the sought-after displacement solution vector

The operation of SSOL reflects a conventional forward reduction and backward substitution operations with specialized treatment of applied displacements. This specialization is compatible with the operation of INV, where the matrix factorization process is modified to ensure the presence of the forward-reduction-compatible terms in the factored matrix. Thus, the INV and SSOL processors exhibit interdependency through both the factored matrix data structure and the specialized treatment of applied displacements.

### 2.4.2 PROCESSOR SYNTAX

This processor follows Testbed command syntax and data management conventions as described in Reference 2.

#### 2.4.2.1 Processor Resets

<i>Argument</i>	<i>Default</i>	<i>Meaning</i>
K	K	First field in dataset name of unfactored matrix; second field in dataset name of factored matrix
KLIB	1	A matrix library
KILIB	1	Library containing factored A
QLIB	1	Library containing applied forces, displacements, solution vectors and reaction forces
EP	1	Flag for residual error calculation
L1	1	First input vector for solution <sup>(1)</sup>
L2	0	Last input vector for solution <sup>(1)</sup>
CON	1	Constant case number (ncon)
REAC	1	Flag for computation of nodal reaction forces
SET	1	Load set number (iset of APPL.FORC.iset.1)
NMAX	0	Number of vectors to process at a time <sup>(1)</sup>
NUFF	0	Scratch library for residual error analysis

<sup>(1)</sup> – calculated by SSOL if required.

### 2.4.3 SUBPROCESSORS AND COMMANDS

Not applicable.

### 2.4.4 PROCESSOR DATA INTERFACE

#### 2.4.4.1 Processor Input Datasets

- JDF1.BTAB.1.8
- APPL.FORC.iset.1
- EQNF.FORC.iset.1
- APPL.MOTI.iset.1
- INV.K.ncon (INV.name.ncon)
- K.SPAR.jdf2 (name.SPAR.jdf2)

#### 2.4.4.2 Processor Output Datasets

- STAT.DISP.iset.ncon
- STAT.REAC.iset.ncon

### 2.4.5 PROCESSOR LOGIC FLOW

Figures 21 through 29 contain logic flowcharts for the major SSOL subroutines including the main subroutine, subroutine DSG0, subroutine DSX (double precision DSXDP) which directs all SSOL functions, subroutine UEVAL (double precision DUEVAL) which directs the forward reduction and backward substitution operations, and subroutines FRWRD and BCKSL (double precision DFRWRD and DBCKSL, respectively) which actually perform the forward reduction and back substitution. Logic for single or double precision factored matrices is separated by subroutine.

The execution of subroutine DSX is divided into two loops over groups of right-hand-side (r.h.s.) vectors. The first loop contains an inner loop to accumulate applied forces and displacements into a single r.h.s. block for each load set (see usage guidelines in §2.4.7). These r.h.s. vector blocks are stored on a scratch library for later use. Subroutine UEVAL is then invoked to perform the solution operation. Nodal reaction forces are calculated in the second r.h.s. vector group loop.

One should note that the factored matrix actually contains partially factored coefficients for off-diagonal terms associated with degrees of freedom at which displacements are prescribed in APPL.MOTI.iset.1. The forward reduction process is carried out for these d.o.f. with the r.h.s. vector containing the value of the prescribed displacement instead of an applied force. The backward substitution process is skipped for prescribed d.o.f.. This procedure results in the correct displacements being calculated for non-prescribed d.o.f. including the effect of internal forces due to the prescribed displacements. The displacement solution vector also receives the correct prescribed displacement values through this process. Static nodal reactions are calculated based on the full solution vector and the unconstrained stiffness matrix.

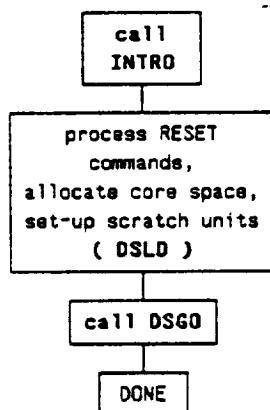


Figure 21 SSOL main program logic flowchart.

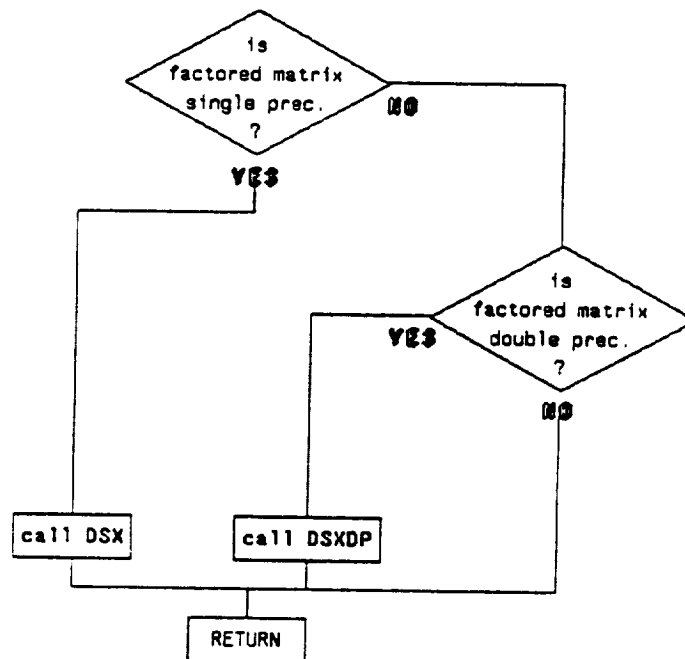


Figure 22 DSGO logic flowchart.

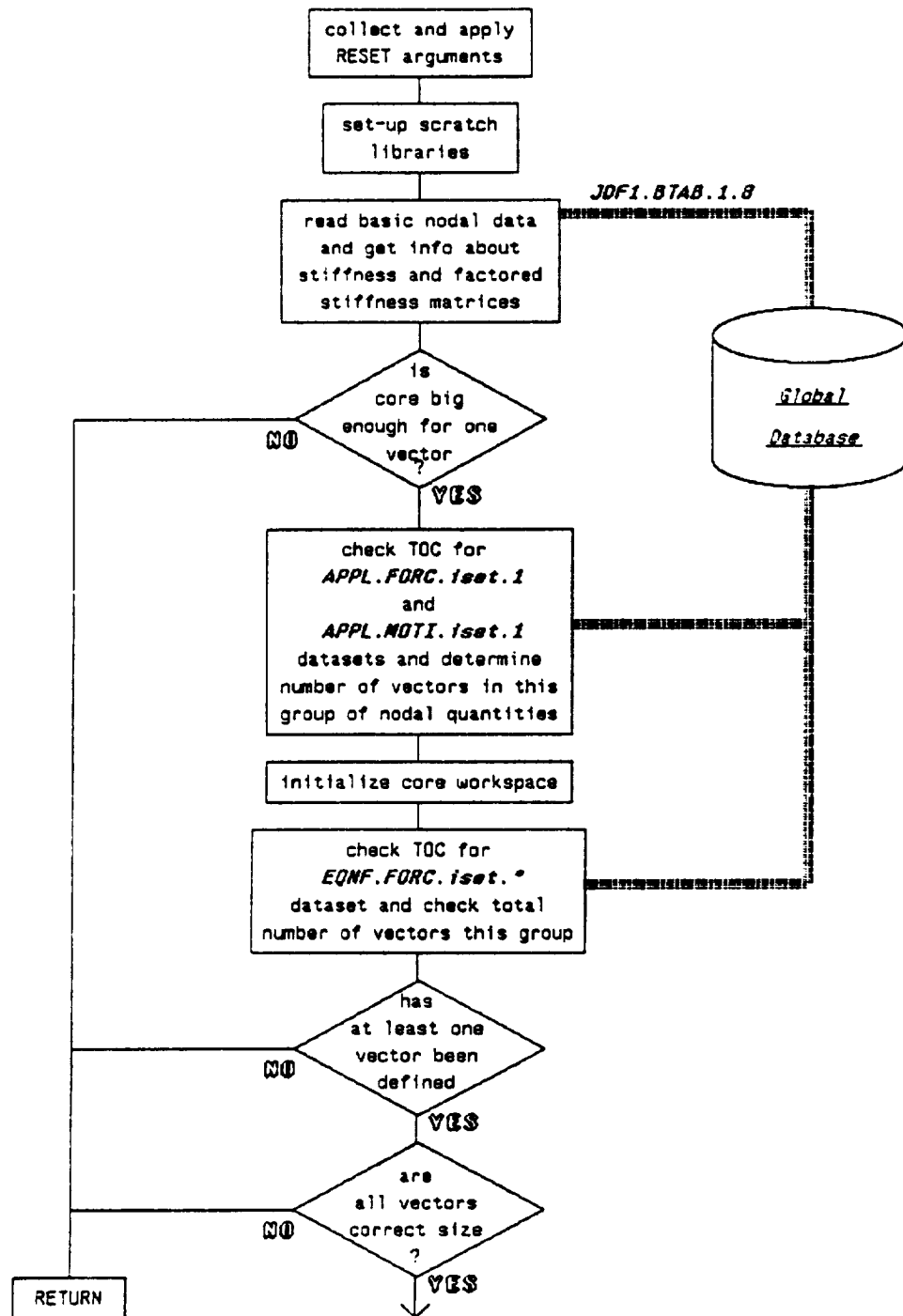


Figure 23 DSLDD logic flowchart.

ORIGINAL FIGURE IS  
OF POOR QUALITY

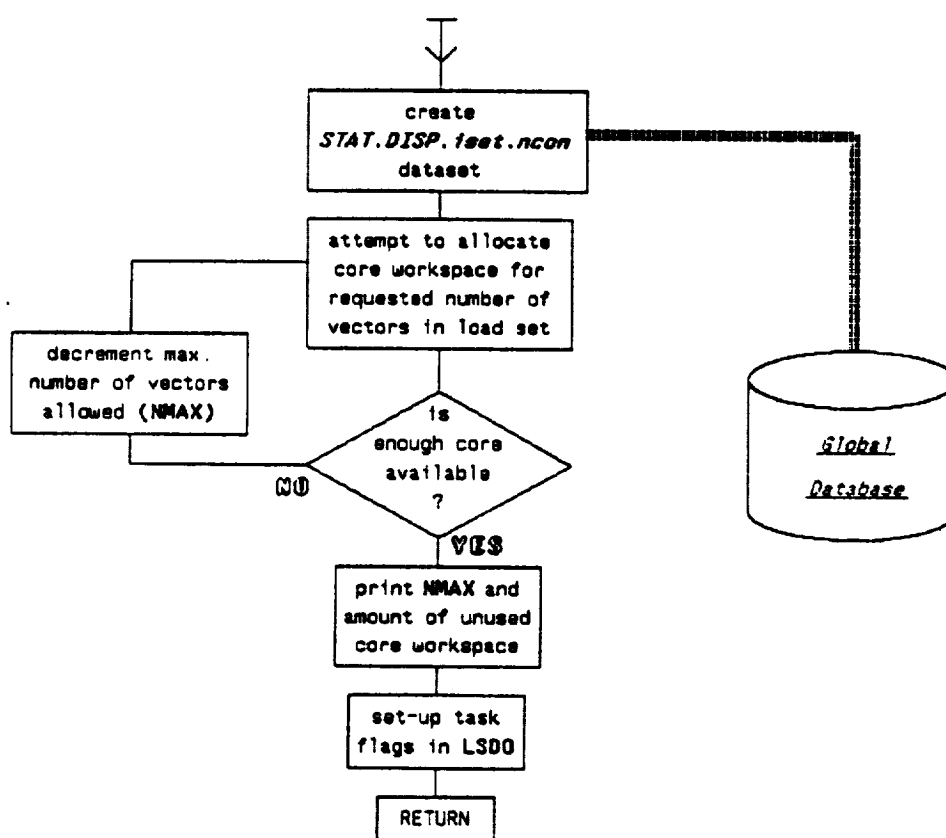


Figure 23 Concluded.

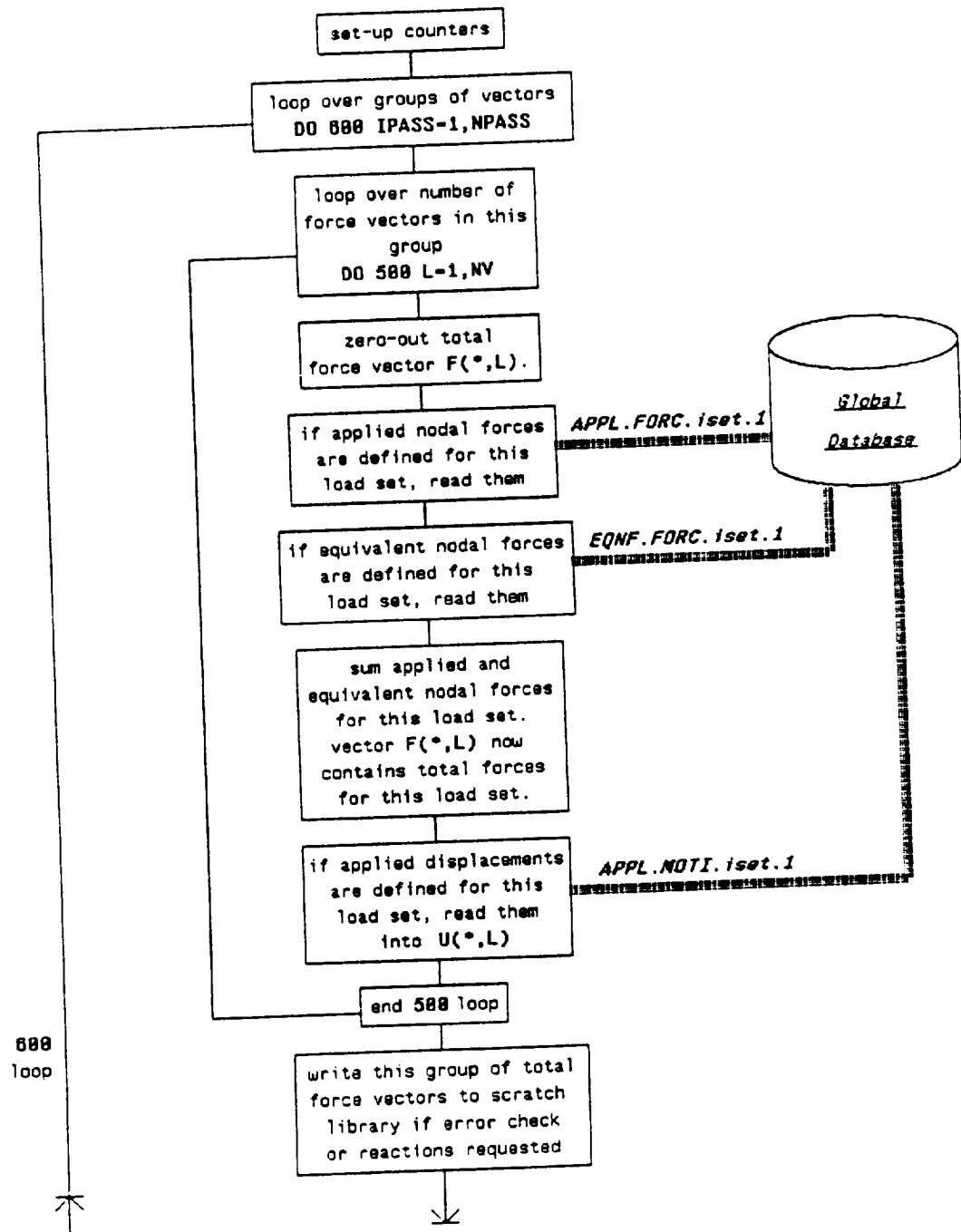


Figure 24 DSX (DSXDP) logic flowchart.

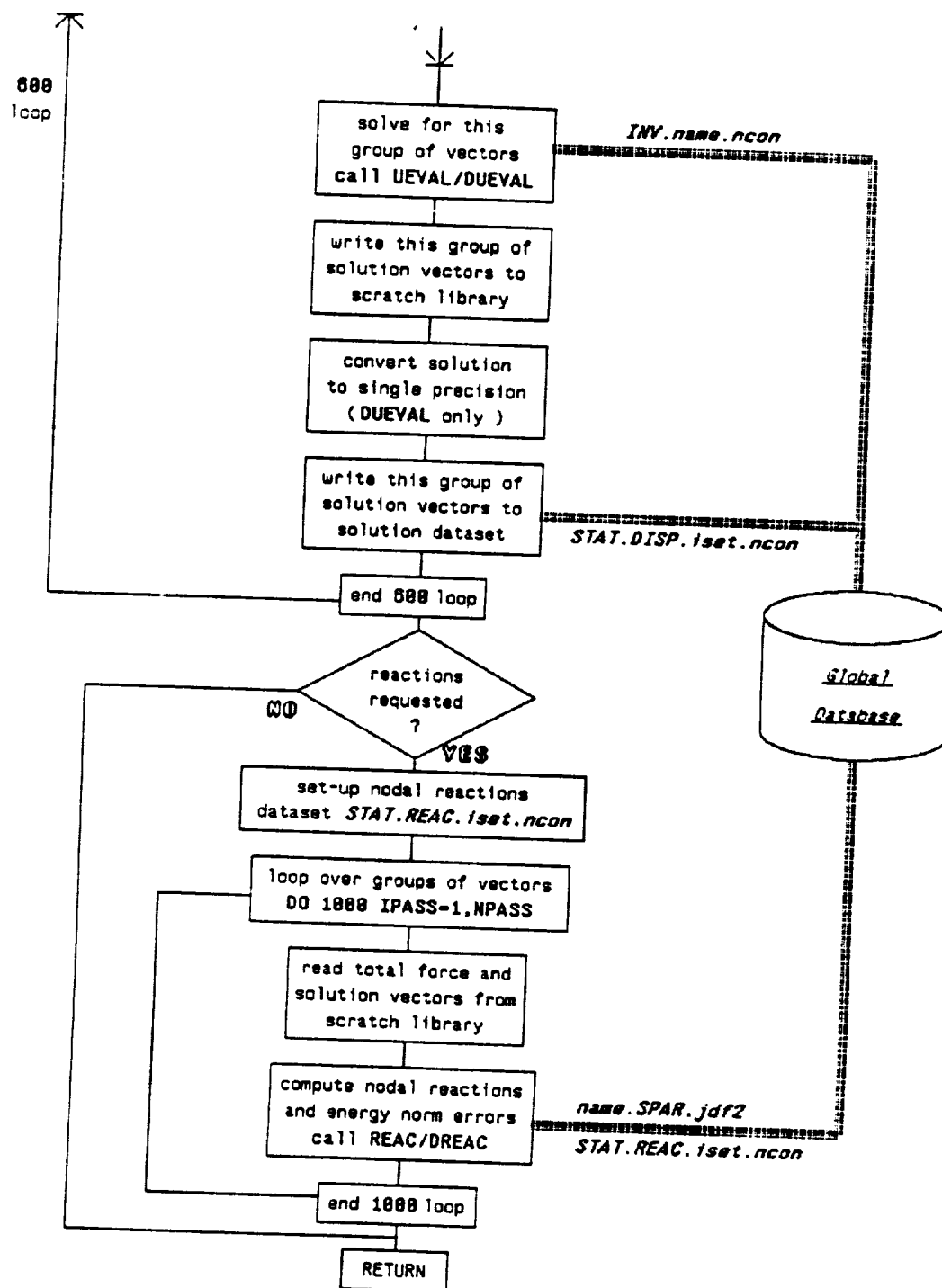


Figure 24 Concluded.



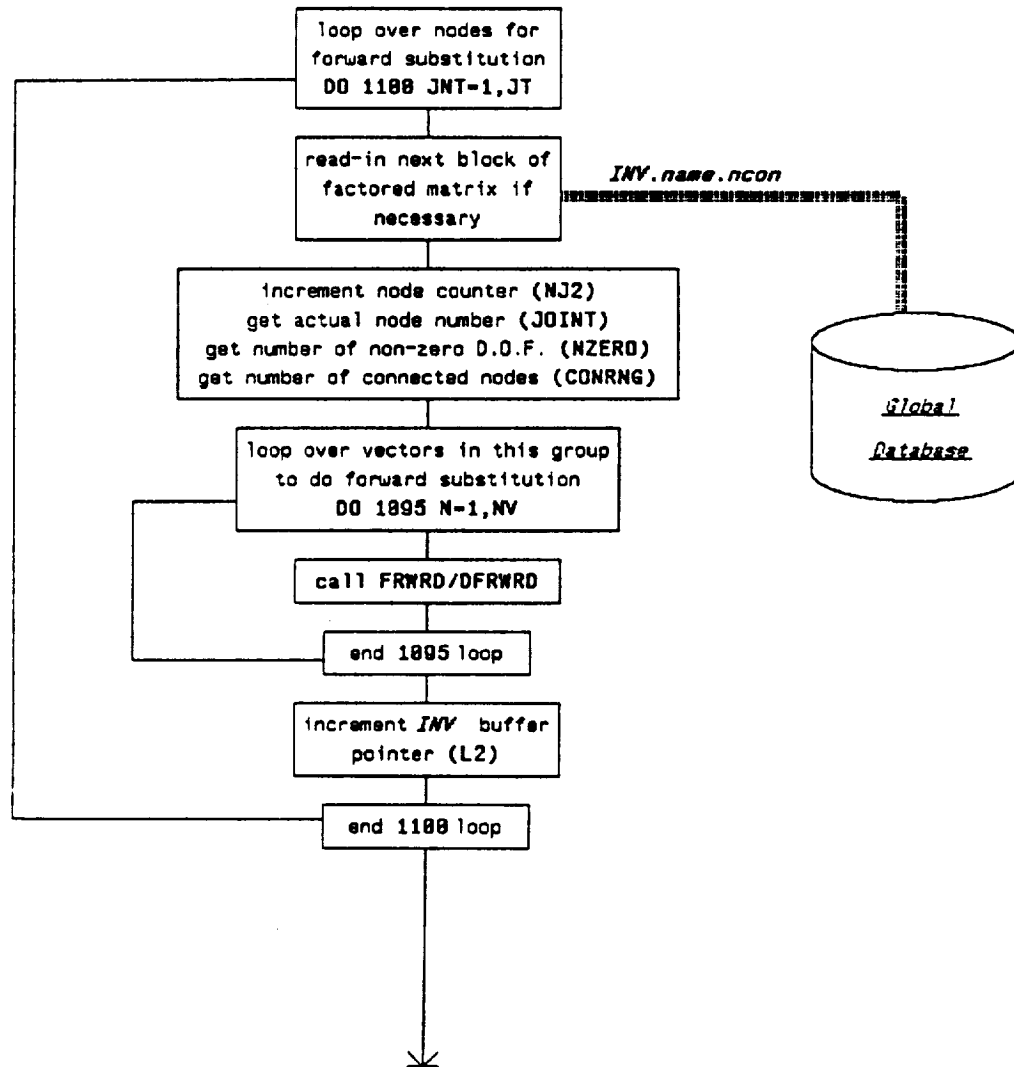


Figure 25 UEVAL (DUEVAL) logic flowchart.

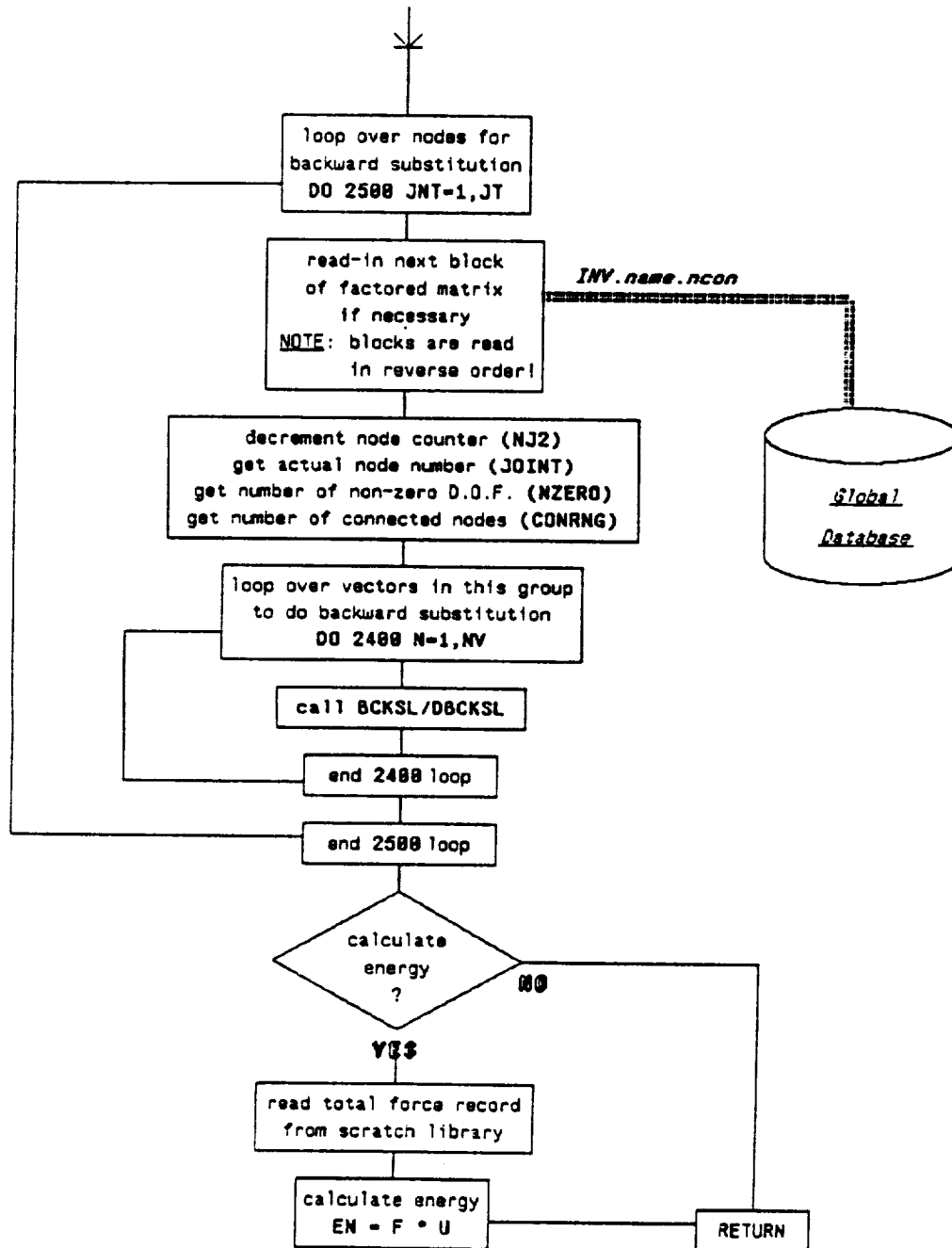


Figure 25 Concluded.

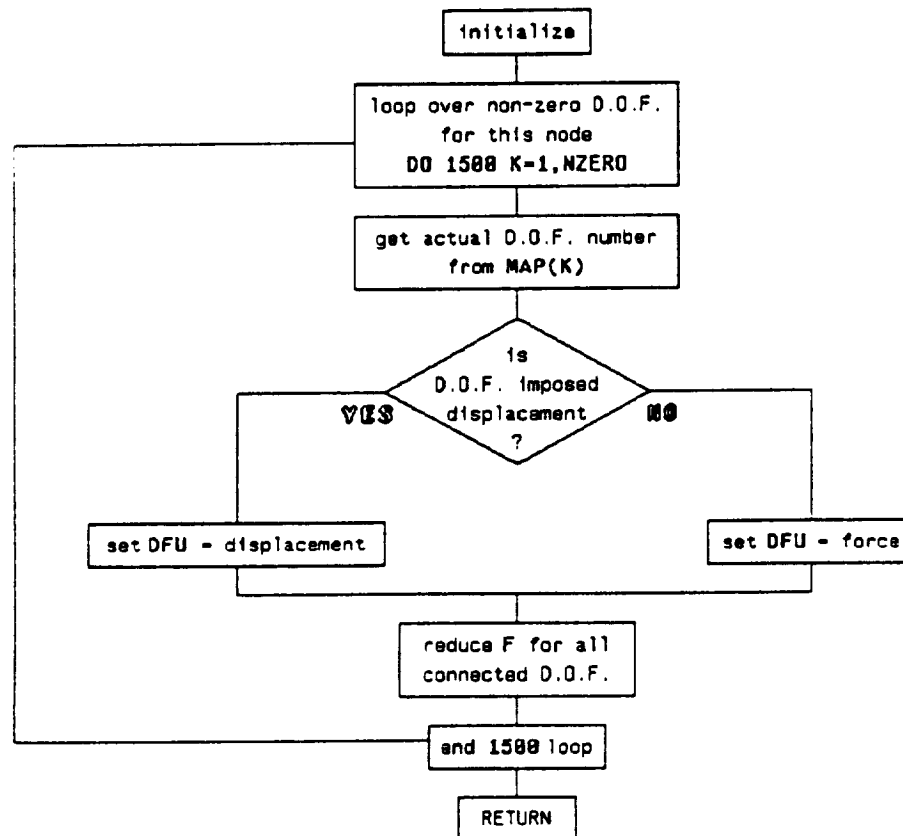


Figure 26 FRWRD (DFRWRD) logic flowchart.

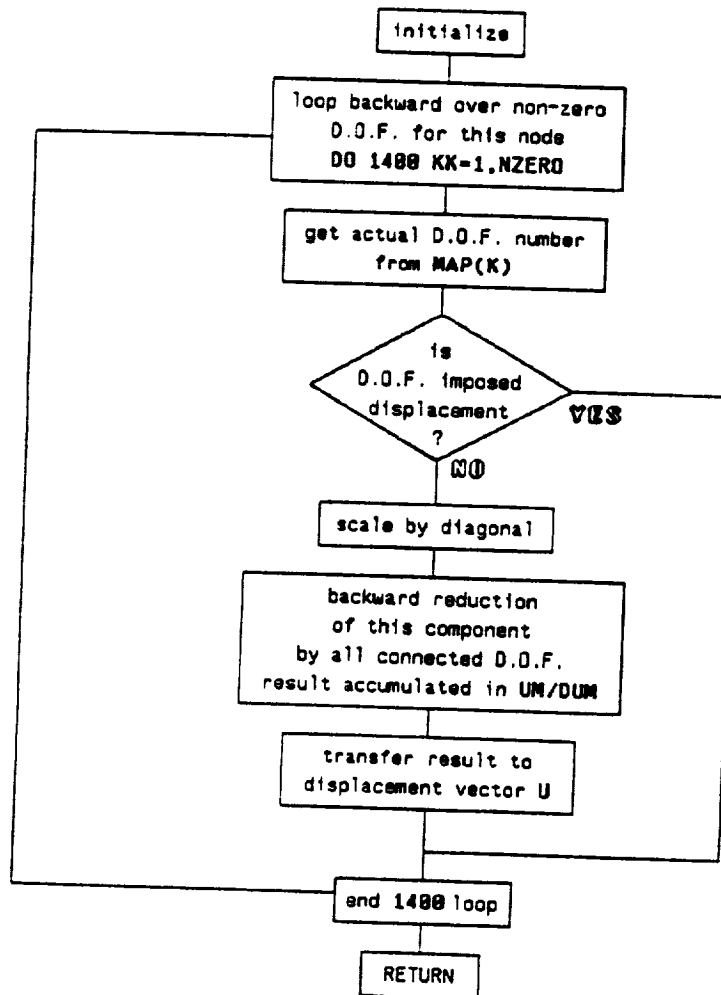


Figure 27 BCKSL (DBCKSL) logic flowchart.

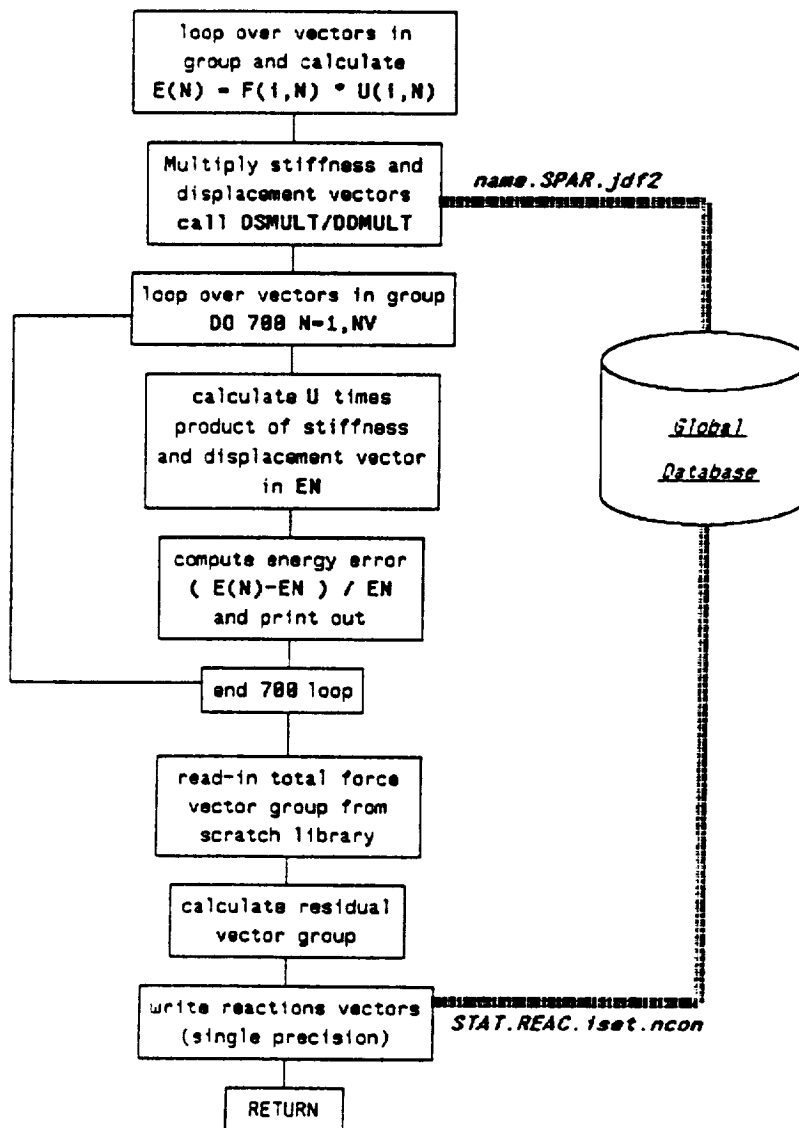


Figure 28 REAC (DREAC) logic flowchart.

ORIGINAL PAGE IS  
OF POOR QUALITY

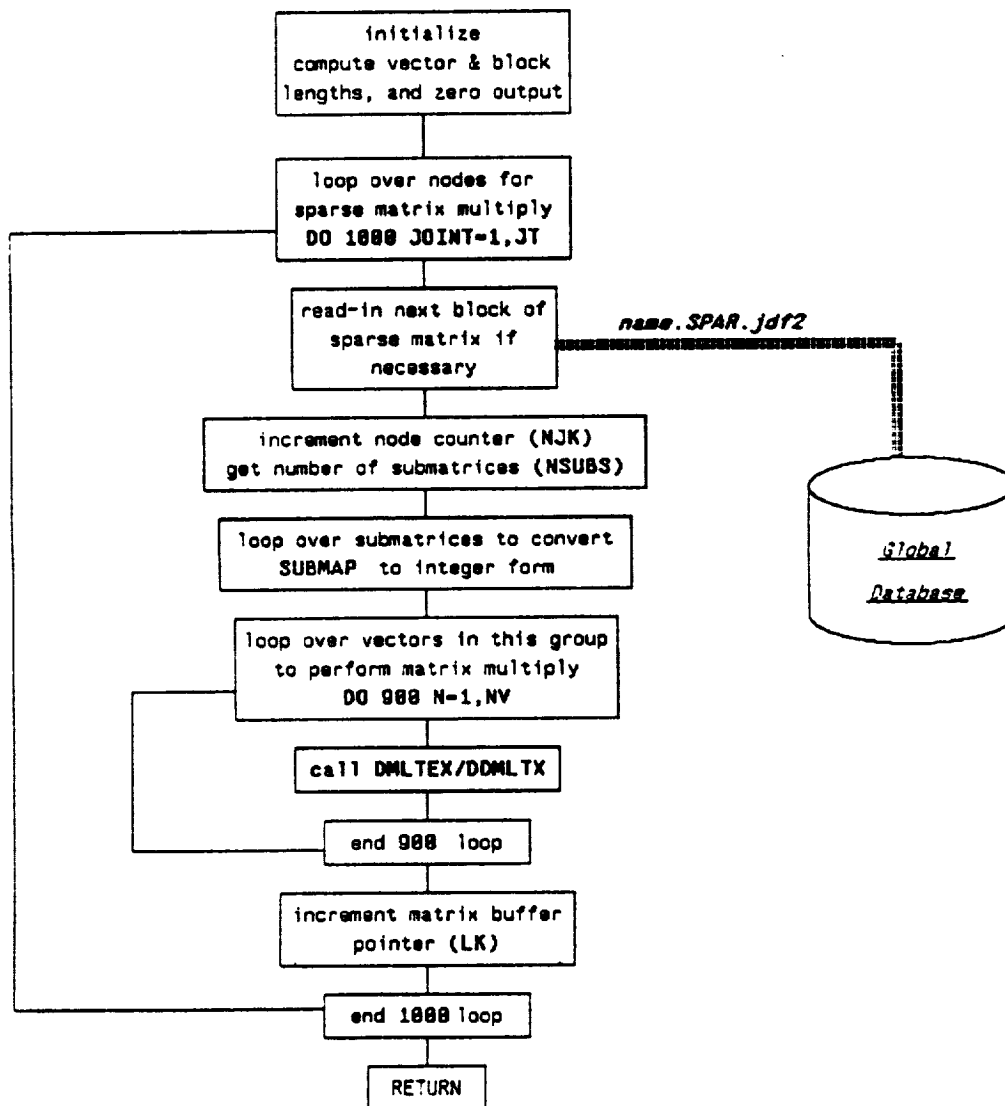


Figure 29 DSMULT (DDMULT) logic flowchart.

### 2.4.6 PROCESSOR DATA FLOW

Core workspace is allocated in subroutine DSLD. Workspace is always allocated for a directory vector of dataset sequence numbers indicating all applied loading vectors to be included in a particular solution process. Beyond this directory vector, workspace is allocated for a vector of task flags indicating which vectors of groups to process, for scratch blocks of assembled right-hand-side vectors and solution vectors, and for a block of either the unfactored or factored system matrix, whichever is larger. Any remaining core workspace is used to store the applied force vectors. If all of these data can fit into the available core space, execution proceeds. If not, the number of right-hand-side vectors to be processed in one pass is decremented and allocation is attempted again. If no vectors can be accommodated in the available workspace along with all other required data, the processor terminates. Pointers calculated in DSLD are stored in common block /DSAD/ and used in the call to DSX in subroutine DSGO.

A scratch data library (NUFF) is used to hold accumulated right-hand-side vectors and solution vectors for energy calculation and residual error checking.

### 2.4.7 SUBROUTINE AND VARIABLE NAME GLOSSARY

<i>Subroutine</i>	<i>Description</i>
BCKSL(DBCKSL)	backward substitution
DMLTEX(DDMLTX)	low-level workhorse for DSMULT(DDMULT)
DSGO	switches on single/double to call DSX or DSXDP
DSLD	startup resets, dataset access, core allocation, etc.
DSMULT(DDMULT)	multiply single-precision (or double-precision) sparse matrix by double-precision vectors
DSX(DSXDP)	main solution driver routine
FRWRD(DFRWRD)	forward substitution
REAC(DREAC)	compute static reactions and force errors
UEVAL(DUEVAL)	solution (forward and back driver)

<i>Variable</i>	<i>Routine(s)</i>	<i>Description</i>
CONRNG	UEVAL	number of connected joints
DFU	FRWRD	total force or specified displacement
E	REAC	dot product of F and U
EN	UEVAL	energy = F * U
EN	REAC	dot product of K*U and U
ERR	REAC	error (E - EN)/EN
F	DSX, REAC	total force vectors
JOINT	UEVAL	actual current joint number
LK	DSMULT	matrix buffer pointer
LSDO	DSLD, UEVAL, DSMULT	vector process flag
MAP	FRWRD, BCKSL	map of nonzero dof at current joint
NJ2	UEVAL	joint counter within current INV block
NJK	DSMULT	joint counter within current block of matrix
NMAX	DSLD	number of vectors processed at once
NPASS	DSLD, DSX	number of passes required to process all vectors
NSUBS	DSMULT	number of connected submatrices
NUFF	DSLD, REAC, DSX	error analysis and reactions scratch library
NZERO	UEVAL, FRWRD, BCKSL	number of nonzero dof at current joint
U	DSX, BCKSL, REAC	solution vectors
UM/DUM	BCKSL	result of back reduction (displacement)

#### 2.4.8 USAGE GUIDELINES AND EXAMPLES

A deficiency was found and corrected in the logic of SSOL relating to the use of prescribed motion components. The deficiency, briefly, is that no check of the APPL.MOTI.iset.1 contents is made to ensure that degrees-of-freedom constrained to be zero in the relevant CON..ncon dataset are indeed zero in the APPL.MOTI.iset.1 dataset. Nonzero values in these positions of APPL.MOTI.iset.1 will cause erroneous results to be calculated in the back-substitution phase of the solution. The existence of an incorrect solution will be obvious to the user upon examination of the force errors in the STAT.REAC.iset.ncon dataset. The moral: *"Always check your reactions!"*



### 3.0 Utility Processors

Various utility processors are available in the CSM Testbed. The utility processor for general matrix and vector arithmetic is called AUS and is described in this Chapter. AUS provides various matrix arithmetic functions as well as subprocessors to construct and modify data tables. Commands to perform the matrix and vector functions are summarized in Table 3.0-1 according to their functional categories. Detailed information about the implementation of such commands is contained in the remainder of this section.

Table 3.0-1 Selected AUS Commands

<u>Command</u>	<u>Command Description</u>
ARANK	Form vector of pointers to rank source dataset in ascending order
DRANK	Form vector of pointers to rank source dataset in descending order
GTOL	Transform joint motion, force, or moment components from global to local joint reference frame
LTOG	Transform joint motion, force, or moment components from local joint to global reference frame
NORM	Normalize a system vector
RECIP	Take reciprocal of single or multiblock dataset <i>i.e.</i> , $z_i = 1./x_i$
RIGID	Define rigid body motion
SQRT	Take square root of single or multiblock dataset <i>i.e.</i> , $z_i = (\text{sign of } x_i) \sqrt{ x_i }$
SQUARE	Square a single or multiblock dataset <i>i.e.</i> , $z_i = x_i^2$
SUM	Add datasets
UNION	Concatenate records of one or more datasets into one dataset

Matrix Multiply Commands

MTRA	Transpose a real, multi-block rectangular matrix
MXTY	Matrix multiply for multi-block datasets: $Z = X^T Y$
MXV	Matrix multiply for multi-block datasets: $Z = XY$
PRODUCT	Matrix multiply: $c_z X c_y Y$ where $c_z$ and $c_y$ are real constants and X and Y are system matrices
RINV	Matrix inverse for a single-block dataset
RPROD	Matrix multiply for single-block datasets: $Z = X^T Y$
RTRAN	Transpose a single-block dataset
XTY	Matrix multiply for multi-block datasets
XTYDIAG	Matrix multiply for multi-block datasets when result is known to be diagonal
XTYSYM	Matrix multiply for multi-block datasets when result is known to be symmetric

### 3.1 Processor AUS

#### 3.1.1 GENERAL DESCRIPTION

- General matrix and vector arithmetic operations
  - Sparse matrix addition (SUM)

$$C = c_1 A + c_2 B$$

A : Sparse or diagonal matrix

B : Sparse or diagonal matrix

C : Sparse or diagonal matrix

- Sparse matrix & system vector multiplication (PROD)

$$z = c_1 c_2 A x$$

A : Sparse or diagonal matrix

x : System vector (SYSVEC format)

z : System vector (SYSVEC format)

- Diagonal matrix result (XTYDIAG)

$$D = x^T y$$

x : Multi-block dataset

y : Multi-block dataset

D : Diagonal matrix result

- Other specialized functions:
  - ARAN, DRAN - Form sorting index
  - LTOG, GTOL - SYSVEC coordinate transformations
  - NORM - Normalize system vector
  - RIGID - Construct rigid motion vectors
- Functions for general vectors:
  - SQRT - Termwise square-root
  - SQUARE - Termwise square
  - RECIP - Termwise reciprocal
  - MXTY, XTY -  $z = x^T y$
  - MXTRAN -  $z = x^T$
  - MXV -  $z = xy$
  - XTYSYM -  $B = x^T y$
  - RINV -  $Z = X^{-1}$

RPROD -  $z = xy$

RTRAN -  $z = x^T$

- o Functions for substructure generation:
  - SSID - Set substructure identifier
  - SSPREP - Prepare substructure definition
  - SSM, SSK - Generate substructure mass, stiffness matrices

### 3.1.2 PROCESSOR SYNTAX

This processor follows Testbed command syntax and data management conventions as described in Reference 2.

#### 3.1.2.1 Processor Resets

None.

### 3.1.3 SUBPROCESSORS AND COMMANDS

<i>Command</i>	<i>Default</i>	<i>Meaning</i>
INLIB	1	Input library (default)
OUTLIB	1	Output library (default)
DEFINE	-	Define correspondence between a symbol name and its library and dataset
MACRO	-	Set values for CLIP macrosymbols from database entities
TABLE	-	Invoke TABLE subprocessor
SYSVEC	-	Invoke SYSVEC subprocessor
ALPHA	-	Invoke ALPHA subprocessor
ELDATA	-	Invoke ELDATA subprocessor

AUS commands such as SUM, PROD, and XTYDIAG take the algebraic form

$$output\_dataset = comand(input\_dataset\_1, input\_dataset\_2)$$

Commands such as SQRT and NORM, which use only one input dataset take the form

$$output\_dataset = comand(input\_dataset\_1)$$

### 3.1.4 PROCESSOR DATA INTERFACE

#### 3.1.4.1 Processor Input Datasets

- JDF1.BTAB.1.8
- User-specified datasets

#### 3.1.4.2 Processor Output Datasets

- User-specified datasets

### 3.1.5 PROCESSOR LOGIC FLOW

Figures 30 through 32 contain flowchart diagrams for the AUS top-level subroutine, subroutine PRP2 and subroutine SSUM. Subroutine AUS controls, in detail, the operation of the AUS processor. Subroutine PRP2 manages the symbolic names of external matrix and vector data, used to indicate the sources of command-argument data and the destinations of the resulting data for all AUS functions. Subroutine SSUM implements the SUM command in AUS and is architecturally similar to subroutine SPROD which implements the PROD function.

Additional subroutines are quite numerous and implement the myriad additional AUS processor functions, not all of which are distinctly defined. With little variation, these additional subroutines retain the same basic architecture employed in SSUM, *i.e.*, each functional subroutine handles its own argument interpretation, local data management, input-output management, and invocation of (generally) lower-level computational routines.

ORIGINAL PAGE IS  
OF POOR QUALITY

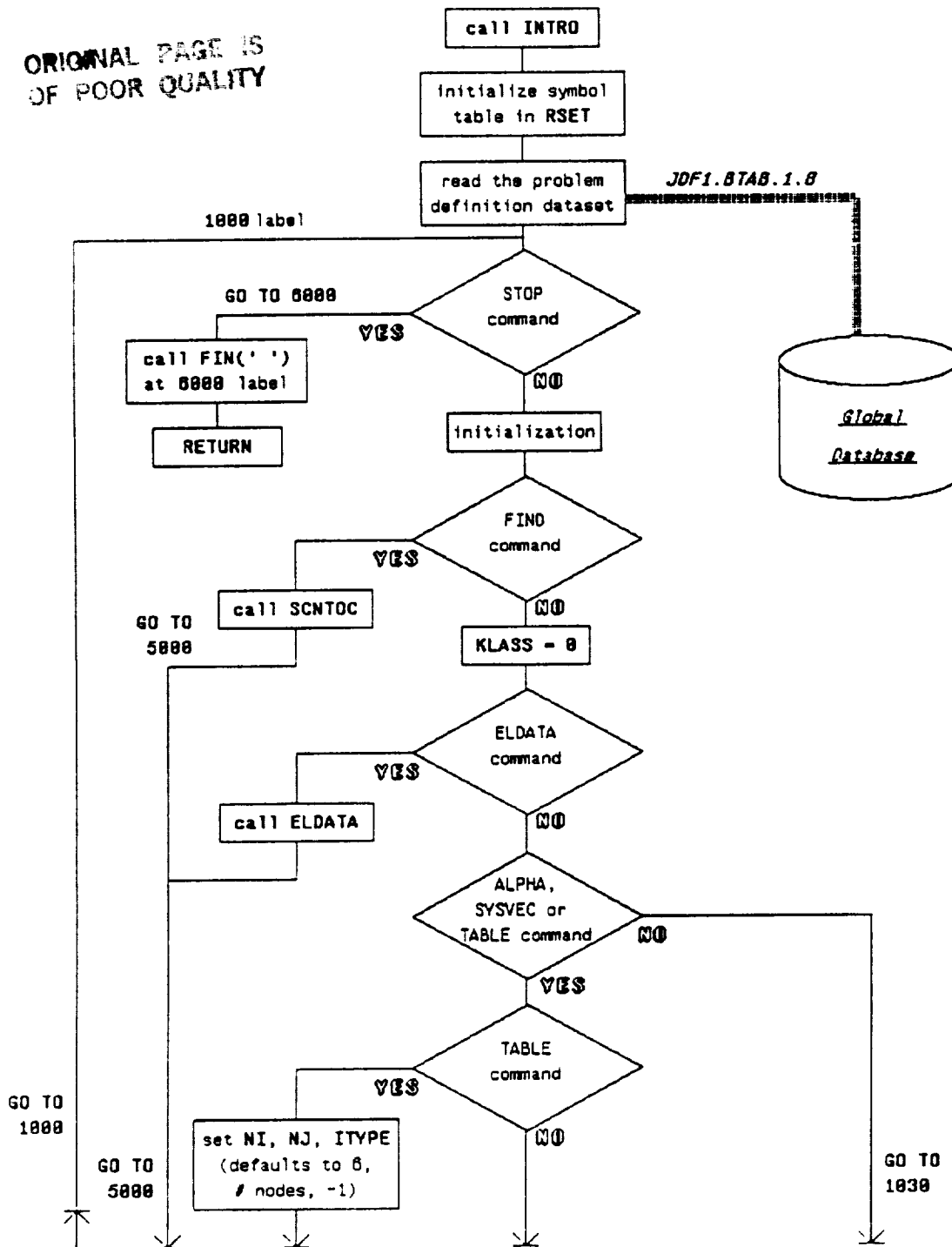


Figure 30 AUS main program logic flowchart.

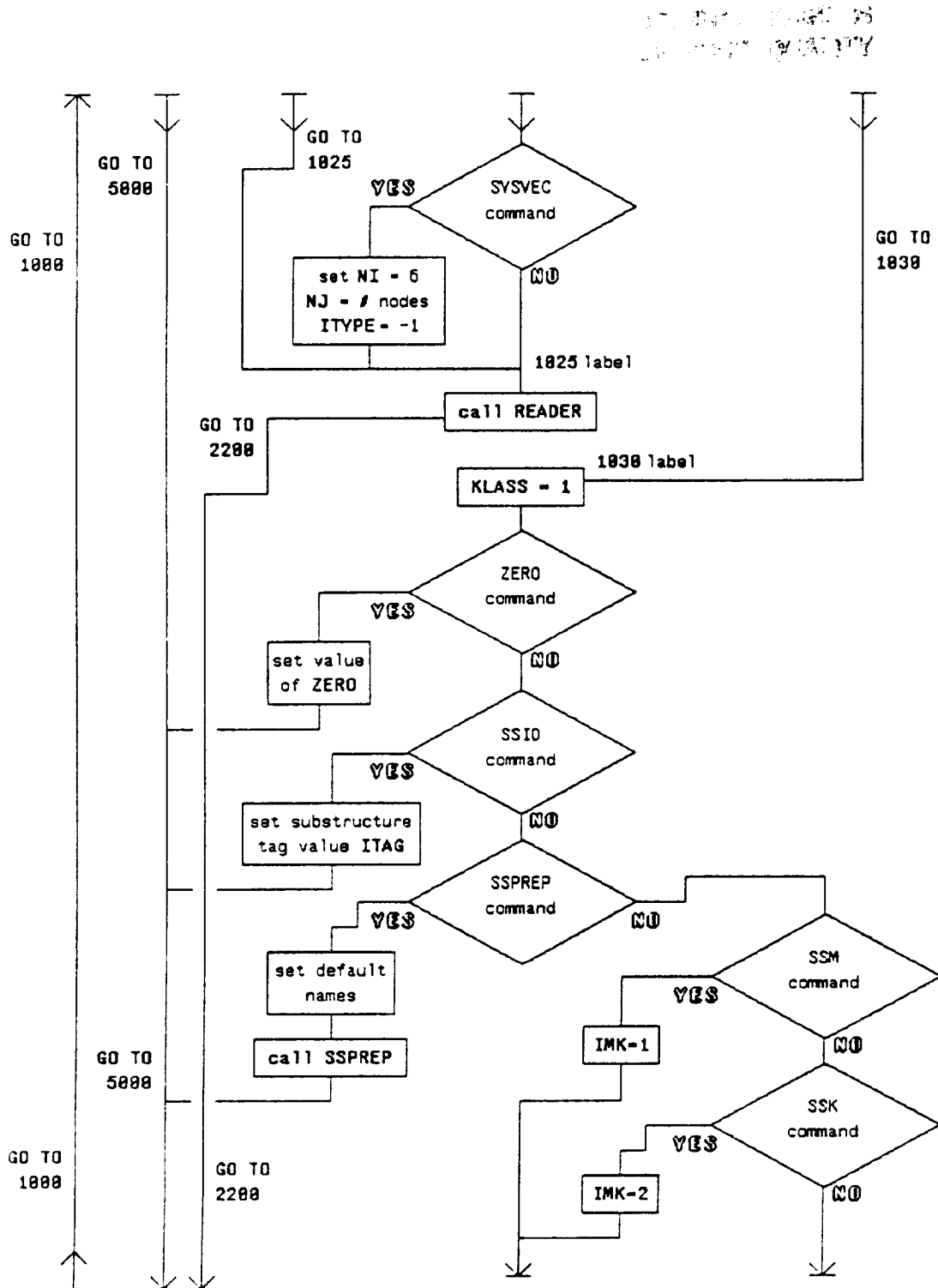


Figure 30 Continued.

ORIGINAL PAGE IS  
OF POOR QUALITY

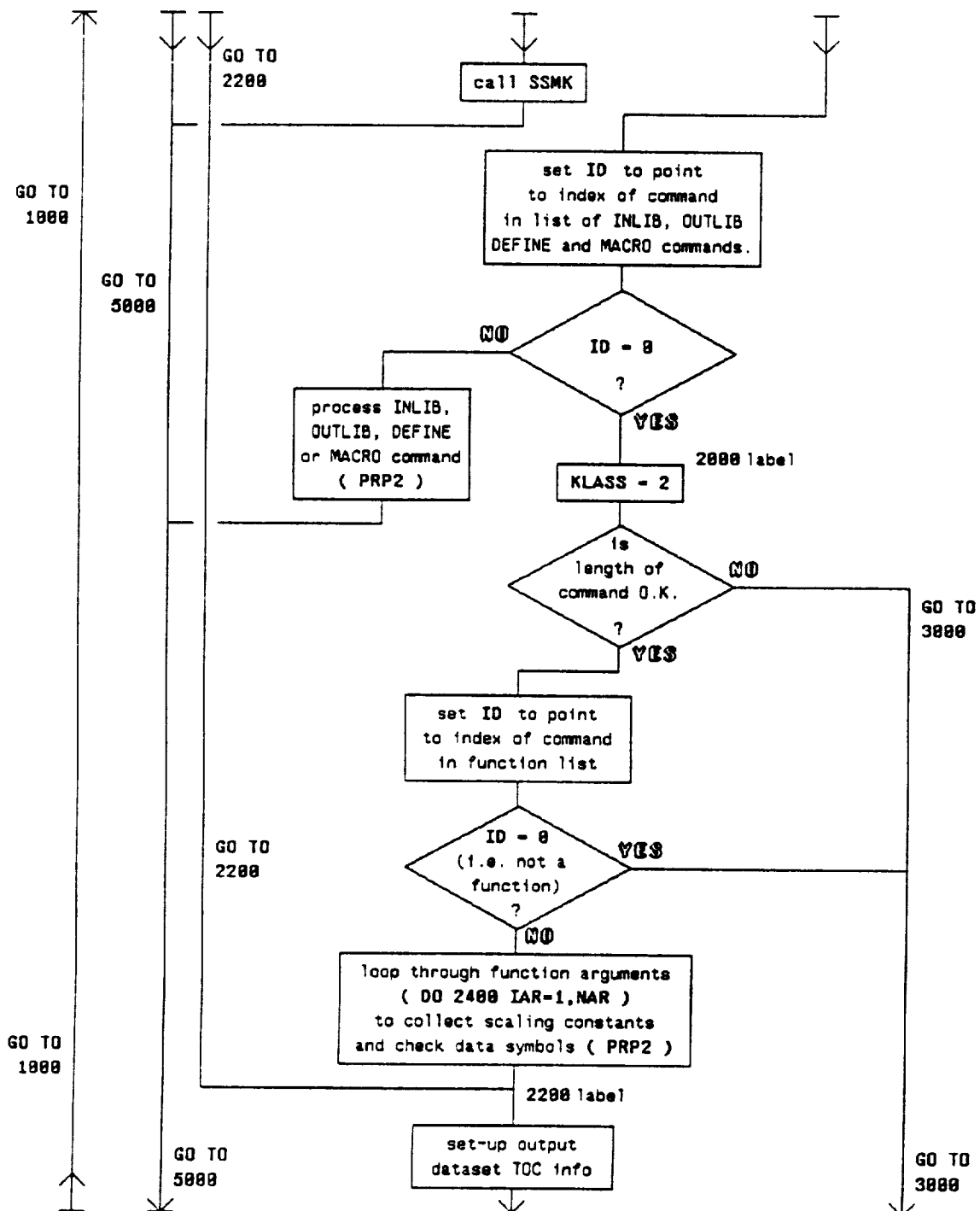


Figure 30 Continued.



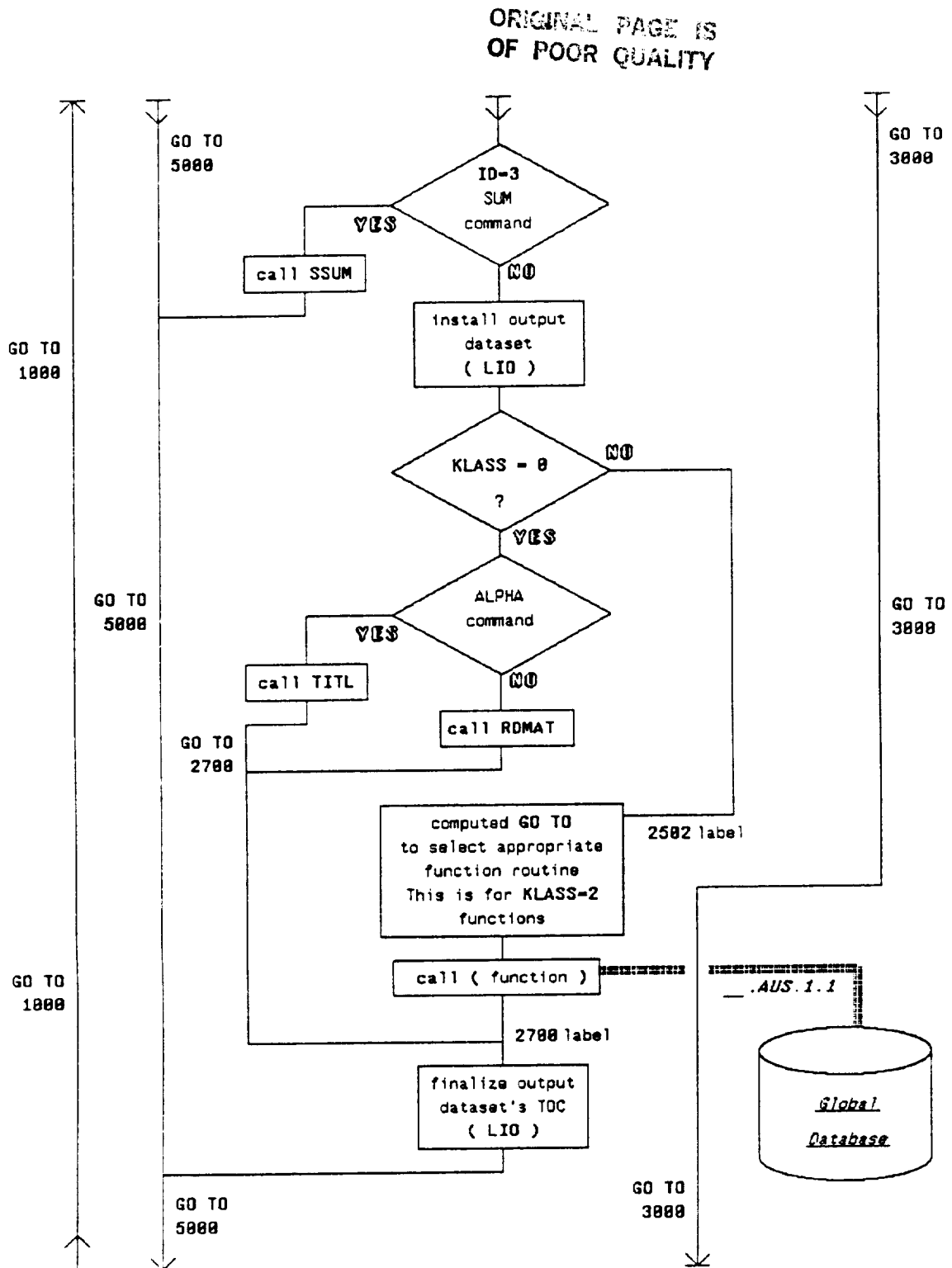


Figure 30 Continued.

ORIGINAL PAGE IS  
OF POOR QUALITY

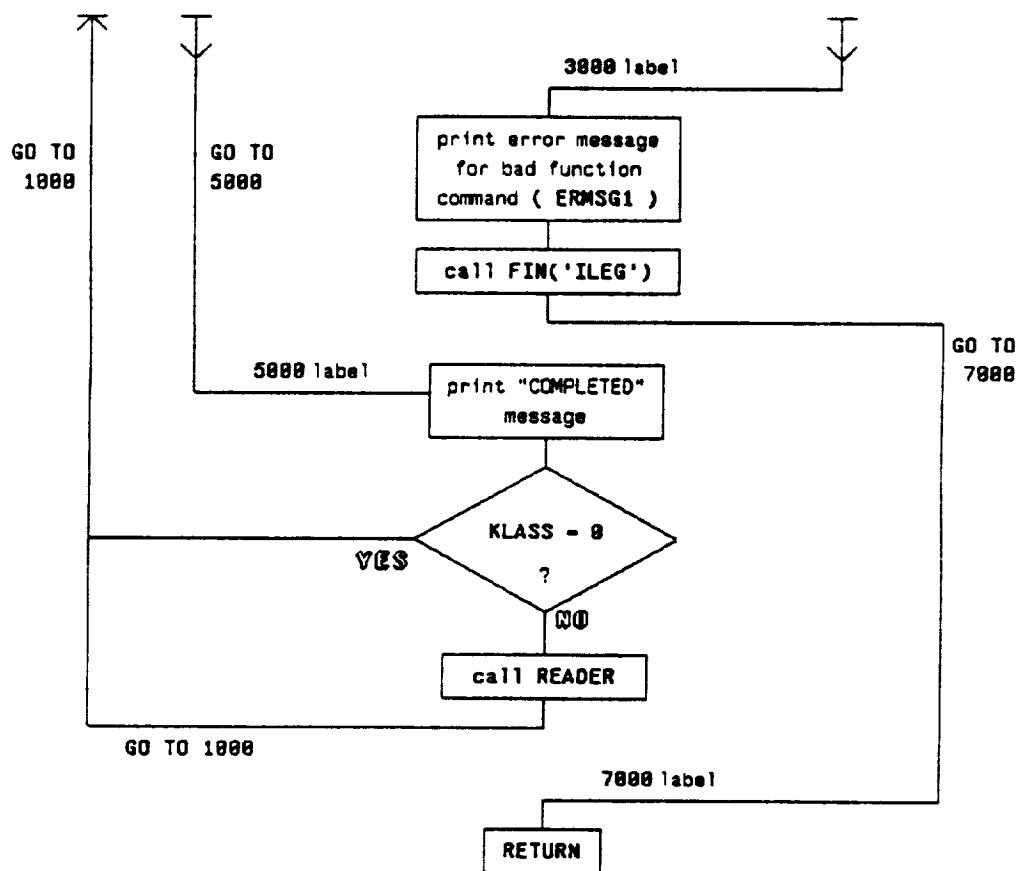


Figure 30 Concluded.

ORIGINAL PAGE IS  
OF POOR QUALITY

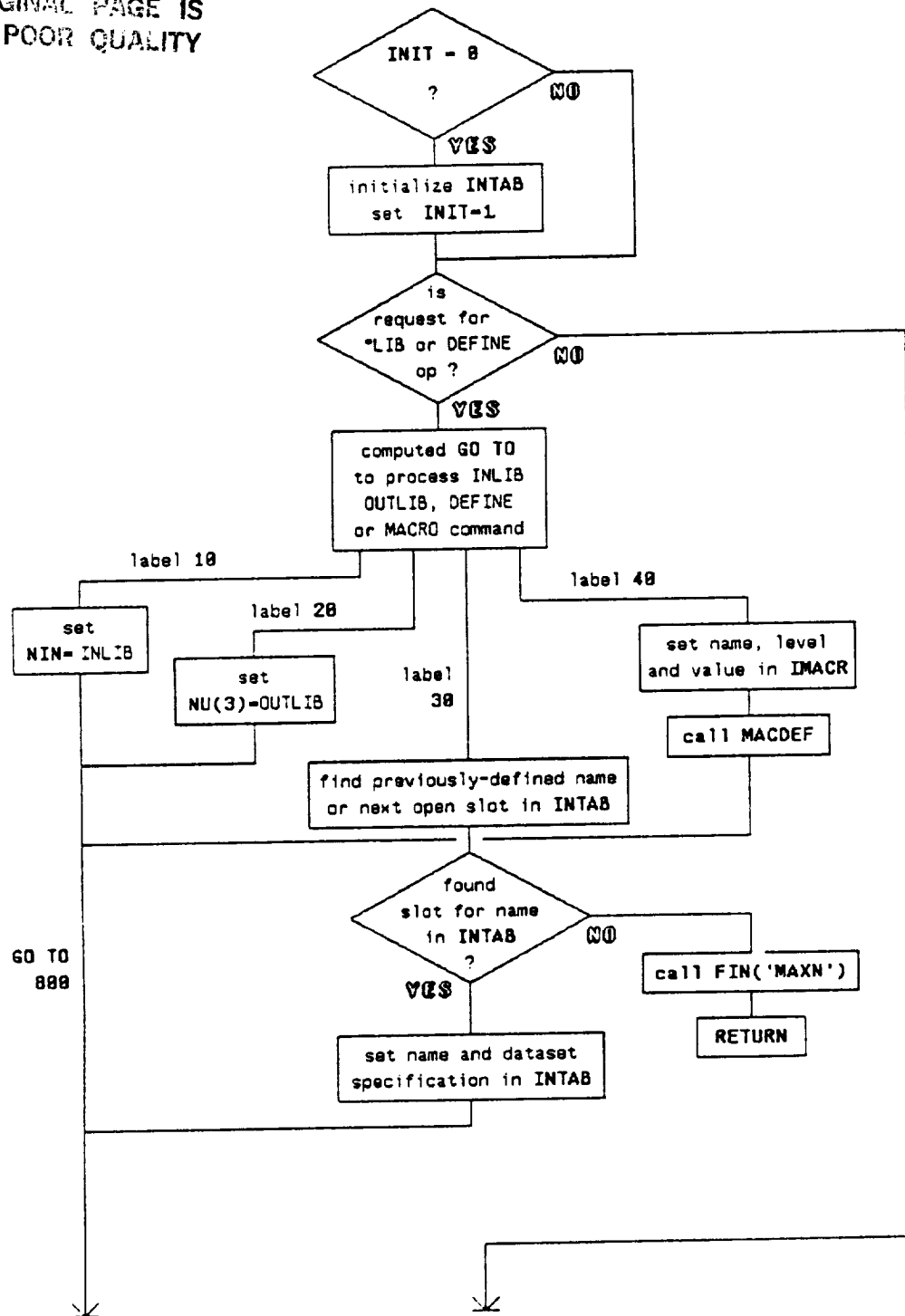


Figure 31 PRP2 logic flowchart.

ORIGINAL PAGE IS  
OF POOR QUALITY

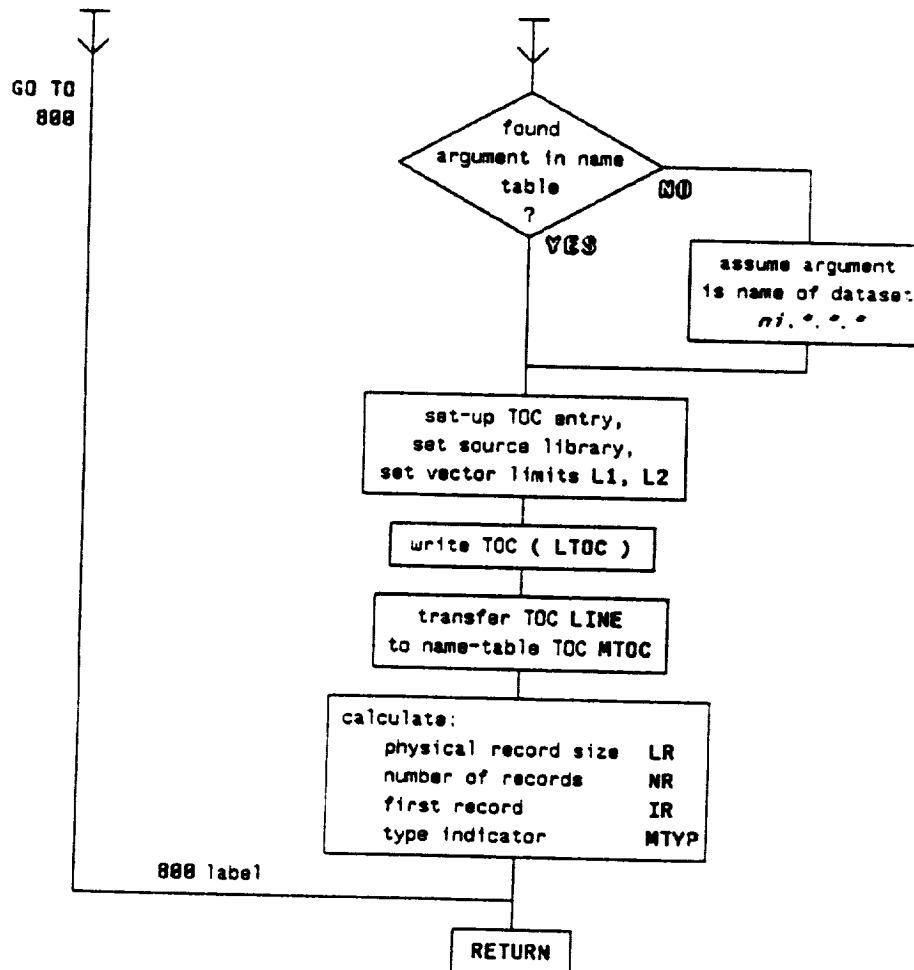


Figure 31 Concluded.

ORIGINAL PAGE IS  
OF POOR QUALITY

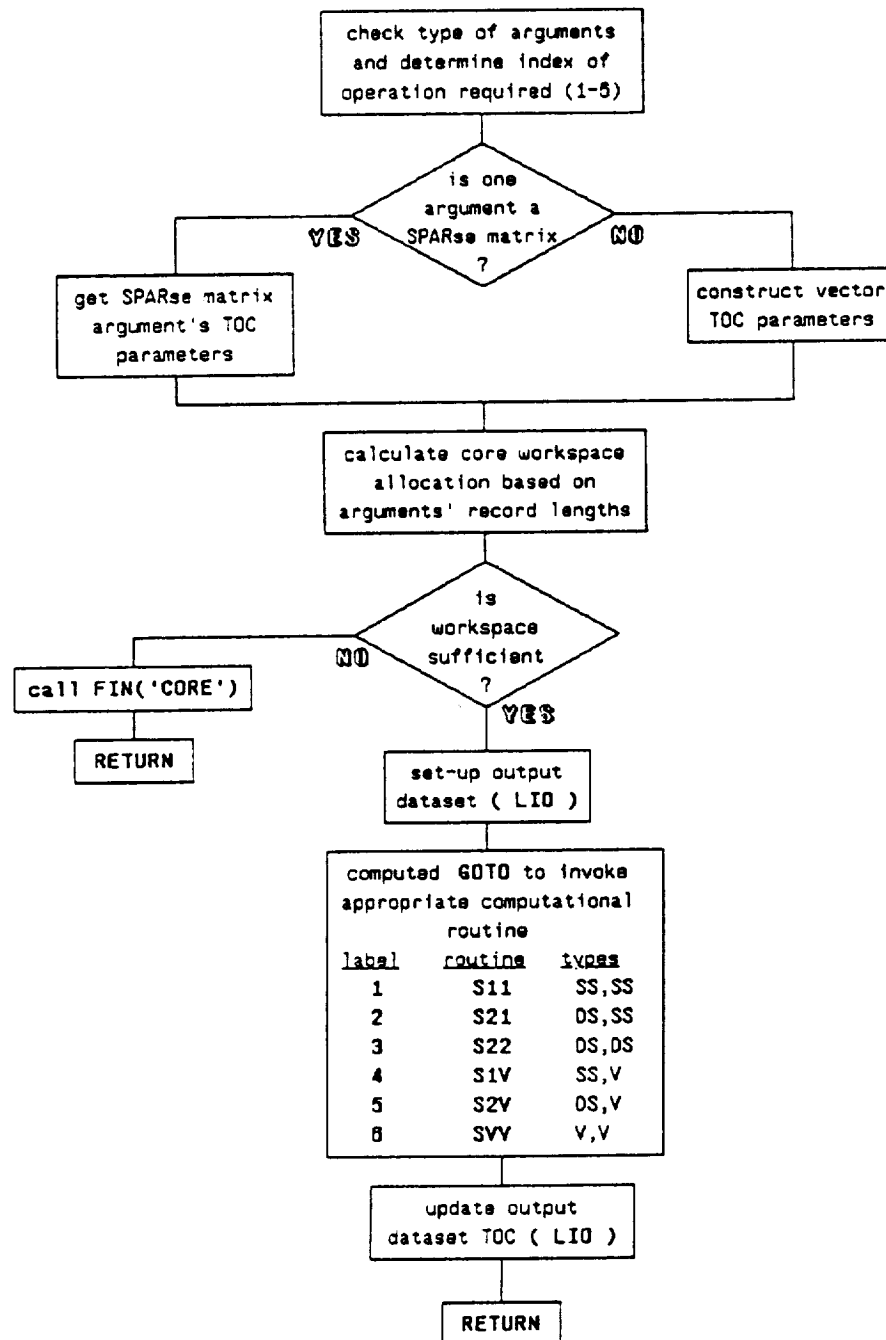


Figure 32 SSUM logic flowchart.

### 3.1.6 PROCESSOR DATA FLOW

Internal data flow in AUS is quite straightforward. Core workspace is always allocated separately by each function, starting from the first word of KA, according to the specific needs of the function. For system matrix operations, space is allocated for one block of the system matrix and the operation proceeds in a block-by-block manner, regardless of the matrix blocksize or the amount of unused workspace.

### 3.1.7 SUBROUTINE AND VARIABLE NAME GLOSSARY

<i>Subroutine</i>	<i>Description</i>
AUS	main level routine resets, commands, etc.
ELDATA	input element data
MACDEF	define CLIP macrosymbol
PRP2	process INLIB, OUTLIB, DEFINE, or MACRO commands or get dataset names
S11	low-level sum of two single-precision sparse matrices
S21	low-level sum of one single-precision and one double-precision sparse matrices
S22	low-level sum of two double-precision sparse matrices
S1V	low-level sum of one single-precision sparse matrix and one single-precision diagonal matrix
S2V	low-level sum of one double-precision sparse matrix and one single-precision diagonal matrix
SVV	low-level sum of two single-precision rectangular matrices
SCNTOC	implementation of FIND command
SSMK	implementation of SSM, SSK commands
SSPREP	implementation of SSPREP command
SPROD	implementation of sparse matrix multiply function
SSUM	implementation of SUM command
TITL	implementation of ALPHA command

<i>Variable</i>	<i>Routine(s)</i>	<i>Description</i>
ERMSG1	AUS	error message
ID	AUS	index of command in function list
IMACR	PRP2	macro definition
IMK	AUS	1 for SSM, 2 for SSK
INIT	PRP2	flag to initialize symbol table
INTAB	PRP2	symbol table
IR	PRP2	1
ITAG	AUS	substructure identifier
ITYPE	AUS	type of data for TABLE command
KA	all	blank common workspace
KLASS	AUS	class of function (0, 1, or 2)
L1	PRP2	first vector to process
L2	PRP2	last vector to process
LR	PRP2	record size
MTOC	PRP2	current TOC line
MTYP	PRP2	1 or 2 for sparse matrices, 3 for other matrices
NAR	AUS	number of function arguments
NIN	PRP2	default input library
NR	PRP2	number of records
NU	PRP2	vector of libraries for arguments and output
ZERO	AUS	zero test parameter

### 3.1.8 USAGE GUIDELINES AND EXAMPLES

The functionality of AUS is quite broad in scope. Indeed, AUS capabilities are true to the notion of "utility." Individual subprocessors are quite powerful, especially in the hands of an experienced user. Arithmetic operations are invoked through a uniform functional command format. The merging of AUS and a higher-level procedural language like CLIP provides a powerful facility for performing iterative simulation algorithms.

One confusing aspect of the AUS functional interface is the dependence of function specifications on argument datatypes. The most egregious instance of such ambiguity is the distinction between the PROD, RPROD and MXV functions; all perform matrix-vector multiplications, but each for a different matrix data storage structure. On the other hand, the SUM function handles data structure details transparently.

All functions provided by AUS operate on either one (*e.g.*, Sqrt, NORM) or two (*e.g.*, SUM, PROD) arguments. Evaluation of general algorithmic expressions involving more than one arithmetic operation or function must be executed in succession. The user must provide scratch workspace in the correct form. Thus, the DEFINE function may be frequently invoked. No facility, aside from the data library Table of Contents, is provided for saving data symbol definitions for later use. Table of Contents data are also used to define parameters like length and number of vectors associated with a data symbol or name. Without exception, vectors used in AUS *must* conform to the SYSVEC or blocked-matrix

(NI by NJ) format. Thus, the linear algebraic characteristics of the vector are intimately tied to its external database representation.

AUS was envisioned to serve as a arithmetic utility for manipulating matrices and vectors associated with a parent finite-element model since the JDF1.BTAB.1.8 dataset is always required, regardless of the user's (as yet unspecified) purpose in invoking AUS. The generality of AUS is compromised by such a restriction, even though many functions operate on or produce system matrix and vector datasets exclusively. Furthermore, system matrix operations are restricted to a single, model-intrinsic topology.

Upon thorough, albeit arduous, examination of the true-to-code AUS subroutine flowchart, one should be convinced that AUS, like virtually *all* SPAR logic subroutines, is not coded in a structured manner. Furthermore, and again "as usual," no extensions to FORTRAN-IV are employed. These two factors combine to make AUS difficult to modify to accommodate increased functionality and/or ease of use.

All logic in AUS is not archaic and unstructured, however. In particular, the notion of a functional *class*, as denoted by the KLASS flag, is very useful for ranking the relative priority and scope of certain groups of functions. The KLASS=0 functions (ELDATA, ALPHA, SYSVEC and TABLE) are used for direct data input to useful external data structures. Numerical and automated data-generation tasks are handled by the specific subprocessors. The KLASS=1 functions (ZERO, INLIB, OUTLIB, DEFINE and MACRO) are used to set administrative constants and information. The MACRO function probably belongs better under KLASS=0, however. One disturbing aspect is that all substructure generation commands are lumped under the KLASS=1 category. The reason for this choice is not obvious. The KLASS=2 category comprises the real workers. These are the true arithmetic functions (with minor exceptions for specialized functions like RIGID).

If the class hierarchy in AUS had been rigorously formulated and followed, a highly structured version might have evolved. The structure would have as its centerpiece the separation of classes of functions through the use of class-specific cover subroutines. Such a structure would have enabled a cleaner logic flow and straightforward addition of enhancements like a pan-functional local data manager to eliminate repetitive I/O in the KLASS=2 routines.



## 4.0 References

- 4.0-1 Stewart, C. B., Compiler: *The Computational Structural Mechanics Testbed Data Library Description*, NASA TM-100645, 1988.
- 4.0-2 Stewart, C. B., Compiler: *The Computational Structural Mechanics Testbed User's Manual*, NASA TM-100644, 1989.

**THIS PAGE LEFT BLANK INTENTIONALLY.**



National Aeronautics and Space Administration

## Report Documentation Page

1. Report No. NASA CR-181742		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle The CSM Testbed Matrix Processors Internal Logic and Dataflow Descriptions				5. Report Date December 1988	
				6. Performing Organization Code	
7. Author(s) Marc E. Regelbrugge and Mary A. Wright				8. Performing Organization Report No.	
9. Performing Organization Name and Address Lockheed Missiles and Space Company, Inc. Research and Development Division 3251 Hanover Street Palo Alto, California 94304				10. Work Unit No. 505-63-01-10	
				11. Contract or Grant No. NAS1-18444	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225				13. Type of Report and Period Covered Contractor Report	
				14. Sponsoring Agency Code	
15. Supplementary Notes Langley Technical Monitor: W. Jefferson Stroud					
16. Abstract This report constitutes the final report for subtask 1 of Task 5 of NASA Contract NAS1-18444, Computational Structural Mechanics (CSM) Research. This report contains a detailed description of the "coded" workings of selected CSM Testbed matrix processors (i.e., TOPO, K, INV, SSOL) and of the arithmetic utility processor AUS. These processors and the current sparse matrix data structures are studied and documented. Items examined include: details of the data structures, interdependence of data structures, data-blocking logic in the data structures, processor data flow and architecture, and processor algorithmic logic flow.					
17. Key Words (Suggested by Authors(s)) Structural analysis software Matrix utilities				18. Distribution Statement Unclassified—Unlimited	
Subject Category 39					
19. Security Classif.(of this report) Unclassified		20. Security Classif.(of this page) Unclassified		21. No. of Pages 96	22. Price A05





